Shell Scripts

Scripts

- A shell script is a program written using shell commands.
- Different shells have different syntaxes.
- To specify bash as the script for your shell, use the following line as the first line in the script.

```
#!/bin/bash
```

- It must be the first line and start in column 1.
- Emacs indentation works best if your script name ends in `.sh`.

hello.sh

```bash
#!/usr/bin/bash

echo 'Hello, world!'
```

Running a Script

- You can run a script like this:

```
bash hello.sh
```

- If you make the script executable, you can run it like this:

```
chmod +x hello.sh
./hello.sh
```

- chmod changes the file permissions and +x means add executable access.

Variables

- Shell variables are given values like this:

```
x=foo
y=10000
```

- To get the value of a shell variable, precede it with a dollar sign.

```
echo $x
```

- Important! There can be no space between the variable name, the equal sign, and the value.

Arithmetic

- Arithmetic is done line this:

```
#!/bin/bash

x=100
y=200
z=5
echo $((x + y * z))
```

- Bash only has integer arithmetic.
- Inside `$((  ))` the `$` in front of variable names is not needed.

Command Line Arguments

- Arguments to the script are stored in the variables $1, $2, $3, ...
- This script adds 2 arguments and prints the result.

```
#!/bin/bash

echo $1 + $2 = $(($1 + $2))
```

- If I run it like this (after making it executable):

```
./add.sh 123 45
```

  the output is

```
123 + 45 = 168
```

  or I can run it like this:

```
bash add.sh 123 45
```

- The arguments 123 and 45 are passed on from bash to the script.

C/Java Style Loops

```
#!/bin/bash

for ((i = 1; i <= 5; i++))
do
    echo $i
done
```

- The double parentheses are required.

Output:

```
> bash count.sh
1
2
3
4
5
```

Command Substitution

- Putting a command inside $( ) takes the output and puts it on the command line.

```
> seq 5
1
2
3
4
5
> echo $(seq 5)
1 2 3 4 5
```

- You can also enclose the command in backquotes.

```
> echo `seq 5`
1 2 3 4 5
```

Python Style Loops (1)

```
#!/bin/bash

for i in 1 2 3 4 5
do
    echo $i
done
```

Output:

```
1
2
3
4
5
```

- The syntax is `for` i in list ...
- Lists are not enclosed in anything and the items are separated by whitespace
- But we don’t want to have to list all the numbers.

Python Style Loops

- We can use command substitution to generate the list.

```
for i in $(seq 5)
do
    echo $i
done
```

Output:

```
1
2
3
4
5
```

For Loops on the Command Line

```
> for i in $(seq 5); do echo $i; done
1
2
3
4
5
```

- The placement of semicolons has to be exactly right.
- The semicolon is used to separate commands on the command line.
- Semicolons can be used on the command line where newlines are used in scripts.

```
> x=1; y=2; echo $((x + y))
3
```

Conditionals

```
#!/bin/bash

for i in $(seq 8)
do
    if (($i % 2 == 0))
    then
        echo $i is even
    else
        echo $i is odd
    fi
done
```

Output:

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
```

For Loops with Files

- Make backup copies of C files.

```
> ls
a.c  b.c  c.c
> for f in *.c; do cp $f $f.bak; done
> ls
a.c  a.c.bak  b.c  b.c.bak  c.c  c.c.bak
```

Case Statement

```
#!/bin/bash

case $1 in
    apple | orange | pear | peach)
      echo fruit
      ;;
    brocolli | cabbage | lettuce)
      echo veg
      ;;
    *)
      echo unknown
      ;;
esac
```

Sample Runs

```
> ./case.sh apple
fruit
> ./case.sh lettuce
veg
> ./case.sh hamburger
unknown
```

While Loops

```
i=0
while ((i < n))
do
  echo $i
  ((i++)
done
```

or

```
i=0
while [[ i -lt n ]]
do
  echo $i
  ((i++))
done
```

Special Variables in Bash

`*`	List of all command line arguments except `$0`
`$0`	Name of the script that's running
`$1` `$2` . . . `$9`	Arguments 1 through 9
`$10` `$11` . . .	Tenth argument, eleventh, . . .
`$#`	Number of arguments
`$?`	Exit status of previous command
PS1	Your prompt

Using Command Line Arguments

```
#!/bin/bash

# Check for a command line argument

if (($# != 1))
then
    echo usage: count.sh n >&2
    exit 1
fi

for ((i = 1; i <= $1; i++))
do
  echo $i;
done
```

- Comments start with # and go to the end of the line.
- `exit 1` exits the program with status 1 (failure)
- >&2 redirects the output to standard error
- $1 is the command line argument

Exit Status

- Every program terminates with an exit status.
- Convention for exit status:
  - 0 means the program succeeded
  - $\neq 0$ means the program failed
- The exit status of a script is the exit status of the last command that was executed
- The `exit` command terminates a script.
  - If no status is given, the script exits with the status of the last command
  - `exit` *n* makes the script exit with status *n*

Environment Variables (1)

- Environment variables are stored separately from shell variables.
- They are inherited by programs, so they can be used to pass information to programs.
- Example: The C compiler (cc) will use any options specified in the environment variable `CFLAGS`. By setting this variable you don't have to specify options every time you use the compiler.
- Setting an environment variable:

```
export VAR=value
```

- This sets both the shell variable and the environment variable.
- When the shell starts, it initializes a shell variable for every environment variable.
- Convention: Environment variable names are all caps.
- Displaying an environment variable: `printenv VAR`
- Displaying all environment variables: `printenv`

Environment Variables (2)

```
> echo $FOO

> FOO=56
> echo $FOO
56
> printenv FOO
> export FOO=56
> echo $FOO
56
> printenv FOO
56
```

Environment Variables (3)

Some important environment variables

PATH	A colon-separated list of directories the shell will search for commands
SHELL	The shell
LANG	The locale
HOME	Your home directory
TERM	Your terminal type
DISPLAY	The X-windows display

Variable Modifiers (Substring)

`${variable:n}`	substring starting at n
`${variable:n:l}`	substring starting at n of length l

- Indexes start at 0.

Example

```
> x=thisisamediumlengthstring
> echo ${x:5:7}
samediu
> echo ${x:5}
samediumlengthstring
```

Variable Modifiers (Length)

`${#variable}`	length of string

- Indexes start at 0.

Example

```
> x=thisisamediumlengthstring
> echo ${#x}
25
```

Variable Modifiers (Remove Prefix)

`${#variable#prefix}`	Remove shortest matching prefix
`${#variable##prefix}`	Remove longest matching prefix

Example

```
> f=foofoo.c
> echo ${f#*foo}
foo.c
> echo ${f##*yfoo}
.c
```

Variable Modifiers (Remove Suffix)

`${#variable%suffix}`	Remove shortest matching suffix
`${#variable%%suffix}`	Remove longest matching suffix

Example

```
f=foobarbar
> echo ${f%bar*}
foobar
> echo ${f%%bar*}
foo
```

Variable Modifiers (Substitution)

`${#variable/pattern/replacement}`	Replace first matching substring
`${#variable//pattern/replacement}`	Replace all matching substrings
`${#variable/#pattern/replacement}`	Replace matching substring at the beginning
`${#variable/%pattern/replacement}`	Replace matching substring at the end

Example

```
> x=abracadabra
> echo ${x/abra/foo}
foocadabra
> echo ${x//abra/foo}
foocadfoo
> echo ${x/#abra/foo}
foocadabra
> echo ${x/%abra/foo}
abracadfoo
```

Variable Modifiers (Example)

- Rename all files with file extension jpeg to the extension jpg:

```
> for f in *.jpeg; do mv $f ${f%jpeg}jpg; done
```

- or

```
> for f in *.jpeg; do mv $f ${f/%jpeg/jpg}; done
```

File Tests

`-e file`	File exists
`-d file`	File is a directory
`-f file`	File is a regular file
`-h file`	File is a symbolic link
`-r file`	File is readable
`-w file`	File is writeable
`-s file`	File is not empty
`-x file`	File is executable

- Used inside `[[ ]]`
- Example: (delete executable files)

```
for f in $* do
  if [[ -x $f ]]
  then
    rm $f
  fi
done
```

- Conditional Operators

&&	and
||	or
!	not

Other Conditional Expressions

- These can be used inside `[[ ]]`
- Comparing strings: `==` `!=` `<` `>`
- Comparing numbers: `-eq` `-ne` `-lt` `-le` `-gt` `-ge`

Example: Print Length of Longest Line

```
#!/bin/bash

longest=0
while read line
do
      if ((${#line} > longest))
      then
          longest=${#line}
      fi
done

echo $longest
```

Example: Number Lines

```
#!/bin/bash

IFS=   # Make read keep whitespace

count=1
while read line
do
    printf "%6d %s\n" $count "$line"
    ((count++))
done
```

Examples: Numbers Lines From Many Files

```
#!/bin/bash

IFS=   # Make read keep whitespace
count=1

function numberfile( ) {
    while read line
    do
        printf "%6d %s\n" $count "$line"
        ((count++))
    done

}

if (($# == 0))
then
    numberfile
else
    for f in $*
    do
        numberfile < $f
    done
fi
```