THE UNIVERSITY OF WESTERN AUSTRALIA

Variables

Lecture 5

Michael J. Wise

Variables

- Like all other computer languages, Bash has variables which store values for later referencing
- Assignment is easy, with one Gotcha:

<variable>=<single value> , e.g.

```
% cat=sherlock
```

NOTE: NO spaces around =

- Evaluating a variable
  - *Put a $ in front of it*

```
% echo $cat  # print to screen (stdout)
Sherlock
% echo cat
cat
```

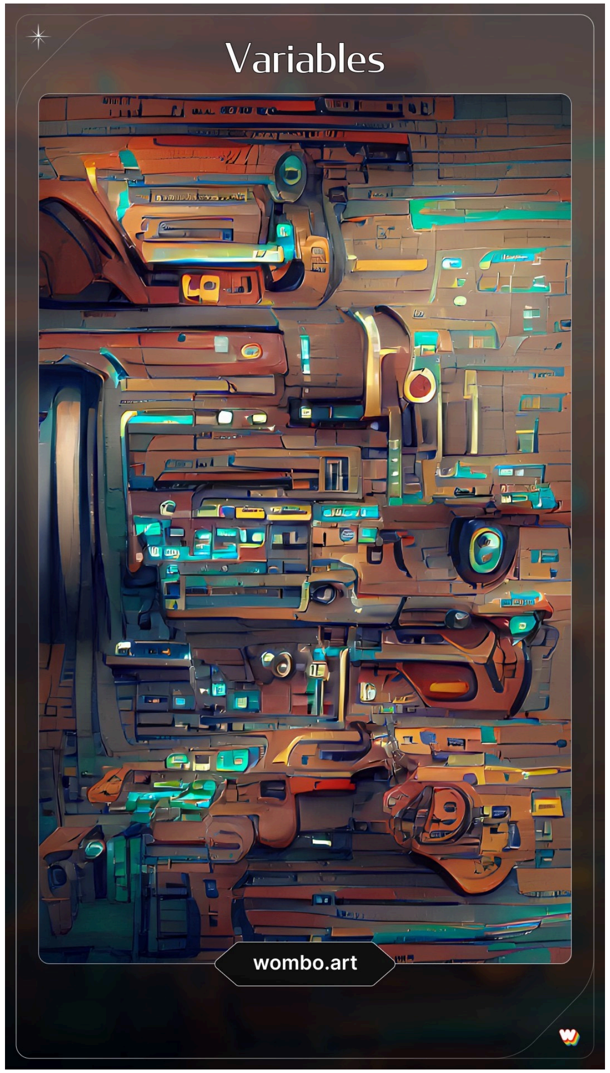
Quoting

```
% a=b c
```

bash: c: command not found

- In other words, to assign more than one thing to a variable: quote them.

```
% a='b c'     # single
               # quotes
```



Quoting/Interpolation

- You can also interpolate the contents of a variable into a string, but for that you need double quotes

```
% echo "The name of my cat is $cat"
The name of my cat is Sherlock
```

- Okay, but what if $ is part of what needs to be printed, e.g. IOU $100 ?
  - *Use back-slash \ to "unspecial" the first $*

```
% owed=100
% echo "IOU \$$owed"
IOU $100
```

- It is sometimes helpful to wrap variable in `{ }`

```
echo "IOU \$${owed}now" # See why?
```

Command Interpolation

- The output of a command can be interpolated into a string, e.g.

```
% echo ”The time now is $(TimeNow)”
```

- Note the round brackets for interpolating output from a program call, versus braces for interpolating contents of a variable

Demo (L0 strikes again)

- In this version of `count_occurences` (where hopefully all the bits now make sense) I want the output to go to a file which is the same as input .txt file, except that the basename (i.e. without `.txt`) has *`_counts`* appended.
- Thus if the input file is `Alice_in_Wonderland.txt`, the output file should be `Alice_in_Wonderland_counts.txt`

Sharing: Where do commands live?

- In Unix, commands (aka executables, binaries) can live in a number of places, public or private
  - `/bin` *top level system commands,* `cat`, `echo`, `ls`
  - `/usr/bin` *system commands that generally come with the system, but may need to be installed,* `gzip`, `make`, `zip`, `gcc`, `python`
  - `/usr/local/bin` *programs that you will have installed for all users of your system locally. In my* `/usr/local/bin` *I have* `hmmsearch`, `pip3`, `ps2pdf`
  - `~/bin` *perhaps you have your own personal directory for programs*
  - `.` *Perhaps the program you need is in the same directory you are in now*

Built-in/System Variables - PATH

- There are a number of variables that are provided by the system. *Environment* variables.
- Perhaps most important of these is PATH
  - *The list of directories the system traverses (in order) to find the named command*
  - *Colon separated list of directories*

```
PATH=~/bin:/usr/local/bin:/usr/bin:/bin:.
```

- Why in that order?
- Including '.' in PATH is controversial
- Including '..' in PATH is to be avoided
  - *Path traversal (injection) attack*

Other Built-in Variables

- `HOME` is your personal root directory (i.e. ~)
- `CD_PATH` short-cuts when doing cd to a named (sub)directory

```
CDPATH=.:~:~/etseq:~/lettuce
```

.profile

- When you start a new window a number of environment variables are set up for you, e.g. `HOME`
- You can also set up some of your own, e.g. `PATH`, in an executable hidden file called `.profile`:

```
#!/bin/bin/env bash
PATH=~/bin:/usr/local/bin:/usr/bin:/bin:.
CDPATH=.:˜:˜/etseq:~/lettuce
PS1=”Now what? ”
umask 077 # makes sure newly opened files
          # have no Group or Other access
          # e.g. rw-------
```

Visibility of variables/export

- The scope of a variable in Bash is just the script it is in.
- Different invocations of the script will have difference instances of the variable, but what about child processes? For example, you want your choice of prompt (PS1) to be used by all child login processes (similarly PATH, etc)

```
export PS1
```

Demo

- `ls` lists the files in a directory. What if I only wanted to print the name of the directory followed by the number of files. So, let’s create `dir_sizes` which, given the path to a directory, reports the number of files in the directory and the total size of the directory and all subdirectories. (For the latter you may care to look at `du`.)

The Bit-Bucket for Unwanted Output

- It can be that your script is more interested in, say, stdout from a command call, but not stderr.
- You can redirect that output to `/dev/null`, known as the Bit Bucket.
- https://en.wikipedia.org/wiki/Null_device

```
some_command > ouput_file 2> /dev/null
```

Literate Programming

- It’s very easy to produce rubbish scripting code: hard to read, hard to maintain.
- Code so that the result can be understood and used by others
- ADD COMMENTS, at least to say
  - *What the script is meant to do*
  - *Summary of the calling arguments*
  - *Authorship*