KUAS
KYOTO UNIVERSITY OF ADVANCED SCIENCE
京都発世界人財

京都先端科学大学

Information Literacy 08

loops and scripts

Ian Piumarta

Faculty of Engineering, KUAS

self-preparation review: `for` loop iterates over a list

a `for` loop sets a variable to each word in a list of words

for each word, the body of the loop is executed

```
for NAME in LIST OF THINGS
do
    ...
    body: one or more
    commands to run with
    NAME set to each item in LIST OF THINGS
    ...
done
```

Q: 1

self-preparation review: ‘ ; ’ can replace newline

semicolons ‘;’ can be used instead of a newlines

```
for NAME in LIST... ; do commands ; ... ; done
```

```
echo one
echo two
echo three

echo one; echo two; echo three
```

Q: 2

self-preparation review: '$' gets a variable's value

`$VARNAME` is the value of variable 'VARNAME'

- assign a new value with `VARNAME=`*`value`*

wildcards are expanded in the *LIST* part of a `for` loop

```
for filename in *.txt; do
  echo $filename
done
```

(indentation is optional but helpful to 'see' the loop)

self-preparation review: loops complicate redirection

"`> output`" first **deletes** the contents of `output`

- use `>` **outside** a loop to capture entire output
- use `>>` **inside** the loop to append output

self-preparation review: `echo` has useful options

`echo -n`	do not to print newline
`echo -e`	expand ‘special’ sequences
`echo -e \n`	output a newline character
`echo -e \t`	output a tab character

braces generate new content

brace expressions list each new item

sequence expressions state the first and last value

both repeat a word with generated content

```
echo {de,re}-{code,classify}
echo {0001..3}-{a..c}
```

- useful for creating directories/files
- useful in `for` loops for processing files

Q: 3 – 8

shell scripts are commands saved in a file

the script acts like a new command

you can write as much as you want inside the script

- complex data management tools
- entire data analysis/processing systems

Q: 9

```
#!/bin/sh
echo hello
```

shell scripts have many ‘pseudo variables’

within a script

- `$1` is the first command-line argument
- `$2` is the second
- and so on

```
#!/bin/sh
echo $1
```

shell scripts have many ‘pseudo variables’

within a script

- `$@` is all the arguments
- `$#` is the number of arguments
- `shift` deletes the first argument (`$1`)

```
#!/bin/sh
first=$1
shift
echo first: $first, count: $#, rest: $@
```

Q: 10