# ULI101 - Introduction to Unix/Linux and the Internet

## Lecture 12 - More Scripting

### Control Structures

#### if-then-elif-else

- to control script according to number of arguments:

```
if [ $# = 0 ]
   then
      echo Usage: cmd argument1 argument2 ... 1>&2
      exit 1
   elif [ $# = 1 ]
      then
         .
         .
   elif [ $# = 2 ]
      then
         .
         .
   else
         .
         .
fi
```

#### for-in

- **for** is used to execute statements for a specifed number of repetitions
- a loop variable takes the values of a specified list, one at a time
- for example, to process a list of strings:

```
for animal in lion tiger bear
do
   echo $animal
done
```

- to process the arguments passed to a script as a list of strings:

```
for var in $*
do
   echo $var
done
```

- to process filenames in a directory, using command substitution:

```
for file in $(ls -a $1)
do
   echo $file
done
```

- or, to process filenames in a directory, including path information:

```
for file in $1/*
do
   echo $file
done
```

#### for

- **for** without the "in" keyword - loop variable takes value of variables $1, $2, $3, etc.

```
for args     # Note that "args" is a user-defined variable
do
   echo $args
done
```

## while

- **while** control structure, loop while test remains true (0 return code)
- to read from the keyboard:

```
while [ "$input" != end ]
do
   printf "OK, give me more: "
   read input
   printf "You typed: '$input'\n"
done
```

- to read from a file:

```
while read input
do
   echo "Input line is: $input"
done < file1
```

- another way to read from a file:

```
cat file1 |
while read input
do
   echo "Input line is: $input"
done
```

- note that the file has to be opened to the **while** loop, not to the **read** statement
- for example, the following would not work, the first line of the file would be printed continuously:

```
while read input < file1
do
   echo "Input line is: $input"
done
```