Intro to Unix Operating Systems

Presented by
Instructor: Greg Scott
Phone: (310)660-3203
email: gscott@elcamino.cc.ca.us
Class Topics

- Intro to Shell Scripting (continued)
  - If statements and decisions
  - Case statements and pattern matching
  - Read command for interaction
  - While loops
  - Until loops
  - For loops

- Commands/Concepts for Class 06
  - break
  - case, continue
  - control chars \c, \n, \t
  - do, done
  - elif, else, esac
  - fi
  - if
  - read
  - test, then
  - while
  - ;;
Intro to Shell Scripts

When a script requires user interaction the writer must ask the question “What if the user does something wrong?”. Then he must decide what actions the script must take in response to the user’s input. For example, what if the user of the greeting script fails to type a number for the sleep time on the command line? If a nonnumeric argument is passed to the sleep command it will produce the error message:

sleep: bad character in argument

Exercise: Run the commands: a) $ greeting hello

b) $ greeting hello 5  c) $ greeting bon jour

Note that in each case, the script encounters an error. We will use the if statement to “decide” when the user has interacted inappropriately with the script.
Intro to Shell Scripts

• With the **if** statement, you can cause certain command lines in your shell script to be executed or skipped over, depending on a given condition.
  – The basic **if** syntax is:
    ```bash
    if test
    then
      command-list1
    fi
    ```
    where *test* takes the form: `[ value1 operator value2 ]` and *command-list1* is one or more commands to be executed if the test is determined to be true. The script executes the next command after **fi** if the test is false.
Intro to Shell Scripts

- The second form of the if is:

  if `test`
  then
    `command-list1`
  else
    `command-list2`
  fi

  If the `test` is determined to be true, `command-list1` is executed while `command-list2` is executed if the `test` is false. Then the script executes the next command after `fi`
Intro to Shell Scripts

• The third form of the if statement is:

```
if test1
then
  command-list1
elif test2
then
  command-list2
else
  command-list3
fi
```

This version gives three or more choices of command-lists that can be executed by using as many elif (else if) groups as needed for additional tests.
Intro to Shell Scripts

- Our first test for the greeting script will be to ensure that the user provides at least two arguments. We can test the value of the $# variable with the statement:
  
  ```bash
  if [ $# -ge 2 ]
  then
    sleep $1
    shift
    banner $*
  else
    echo “Usage: greeting seconds  word(s)”
  fi
  ```
Intro to Shell Scripts

• The –ge is the logical operator for “greater or equal to”. See attachment for details of other test operators.

  Exercise: Copy the file greeting.1. Run the commands:
  a) $ sh greeting.1 hello
  b) $ greeting.1 hello 5

• As you can see, this new version can detect too few args, but does not detect when the first arg is nonnumeric. So we must add testing for a proper numeric first argument for the sleep command. Since the if tests are limited to testing one value against one value, we will use another command, case, which can use pattern matching to test against a range of values.
Intro to Shell Scripts

• A **case** statement is used when you are trying to compare one value against many other values for equality. The general syntax is:

```bash
case value in
  pattern1 ) command-list1;;
  pattern2 ) command-list2;;
  patternk ) command-listk;;
  *) default-list
esac
```

where **value** is compared to each pattern. When a match is found, the corresponding command-list is executed and the case is exited (control moves to the next line after esac). If no match is found, the list of the default case *) is executed.
Intro to Shell Scripts

Each *pattern* is a character string that is compared for equality against *value*. The string may consist of wildcards that are the same as those used for filename generation:

- `*` matches zero or more of any character
- `?` matches exactly one of any character
- `[string]` matches exactly one of any character in *string*
- `![string]` matches exactly one of any character not in *string*
- A given pattern may be one or more separate strings, each separated by a vertical bar (`|`).
Intro to Shell Scripts

• We will use the following case statement in the “then” section of our previous if statement:

```bash
case $1 in
  *[a-z]* ) echo “1st argument must be numeric”;;
  *[A-Z]* ) echo “1st argument must be numeric”;;
  *) sleep $1
    shift
    shift
    banner $*;;
esac
```

Exercise: Copy the file greeting.2. Run the commands:
  a) $ sh greeting.2 hello
  b) $ greeting.2 hello 5
  c) $ sh greeting.2 3 bon jour
• Now, this version is virtually bullet-proof, but it is very unfriendly. Rather than terminating after an error has occurred, we can give the user a chance to provide the proper data by prompting for the value and reading the data from the keyboard into variables using the `read` command.
  – The general syntax is:
    ```
    read var      or      read var1  var2  var3  ...
    ```
    Note: do not place a dollar sign ($) before any of the variables.
Intro to Shell Scripts

• When the read command executes, the following occurs:
  – The script pauses and waits for the user to type a line and press enter.
  – The `read` command reads one line from the `stdin` and assigns values to the named user variables:
  – Each word on the input line is delimited from the next by one or more `whitespace` characters.
  – The first word is assigned to `var1`, the second to `var2`, and so on.
  – If there are more words than `variables`, the leftover words are assigned to the last `variable`.
  – If there are fewer words than `variables`, the leftover `variables` are assigned a null string.
Intro to Shell Scripts

• Prior to the using the read command, we must display a prompting message with the echo command to ask for the correct data. When the message is displayed, we have the option of leaving the cursor on the next line after the message, or next to the message on the same line. For the latter, we double quote the message and place a `\c` (continuation control char) at the end of the text inside the quotes. Some other control characters are `\t` for tab, `\n` for newline, and `\f` for formfeed. See man pages for more control chars.
Intro to Shell Scripts

Example:

```bash
echo "Enter your age? \c"
read age
echo "You entered $age."
```

These three lines produce:

```
Enter your age? 25
You entered 25.
```

Exercise: Copy the file echo0.sh. Run it and enter your age when prompted.

- We will add an echo and read combination to ask the user for and accept a number when the 1st arg is not a number. Replacing the error message with this interaction will make the script seem friendlier! The case is modified as follows:
Intro to Shell Scripts

case $1 in
  *[a-z]*| *[A-Z]* ) echo “Sorry, $1 is not a number.”
    echo "Please enter a time in seconds\c"
    read seconds
    sleep $seconds
    shift
    banner $*;;

...  

Note the use of the | (Or) operator to allow a match for either pattern to execute the commands.

Exercise: Copy the file greeting.3. Run the commands:
  a) $ sh greeting.3  hello 5
  b) $ sh greeting.3  3  bon jour
Intro to Shell Scripts

• Our script has improved greatly, but there are still some holes to fill in. Even though we are prompting for a number to be entered, there is no guarantee that the user will always comply. To make sure that it is entered correctly, we must keep repeating the prompting for the number until the data read in is a number. To repeat steps in a script, we must use a looping statement like `while`, `until`, or `for`. First, we will use the `while` loop.
Intro to Shell Scripts

• The **while** statement causes a command list to repeatedly execute as long as certain conditions remain true. The basic **while** syntax is:

```
while test
  do
    command-list
  done
```

• Note that the **test** syntax and operations are the same as discussed earlier for the if statement. The **do** and **done** are used to indicate the begin and end points of the loop’s body. Although not required, it is conventional to indent command-list to improve the readability of the construct.
Intro to Shell Scripts

- To force the user to enter a number for the sleep seconds we could use:

```bash
loop=y
while [ $loop = "y" ]
  do
    echo "Please enter a time in seconds \c"
    read seconds
    case $seconds in
      *[a-z]* | *[A-Z]* ) continue;;
      *) loop=n ;; esac
  done
```
Intro to Shell Scripts

• Note the use of a special statement “continue” which is used in loops to make the loop return to the top (the while in this case) without executing any of the remaining commands inside the loop. Another special statement for loops is “break” which cause the loop’s execution to stop and the script continues at the next command after the “done”.

Exercise: Copy the file greeting.4. Run the command with $ sh greeting.4 hello 5. When it prompts you for the seconds, be stubborn and enter a couple of erroneous values before you finally give in to the relentless prompts.
Intro to Shell Scripts

• Next, we can prompt the user for data when they fail to provide any args to the script. We replace the else with elif and test for $# = 0. When this occurs we can prompt for the sleep seconds and the banner word using an error trapping loop for the seconds just like in the previous case. Since banner can display anything, there is no need to use any error trapping logic for its input value.

• This new elif section will look like the following:
Intro to Shell Scripts

elif [ $# = 0 ]
then
    loop=y
    while [ $loop = "y" ]
    do
        echo "Please enter a time in seconds \c"
        read seconds
        case $seconds in
            *[a-z]* | *[A-Z]* ) continue;;
            *) loop=n ;;
        esac
    done
    echo "Please enter a greeting"
    read word
    sleep $seconds
    banner $word
Intro to Shell Scripts

Exercise: Copy the file greeting.5. Run the command with: $ sh greeting.5. When it prompts you for the seconds, be stubborn and enter a couple of erroneous values before you finally give in to the relentless prompts. Then enter words for the banner.

Exercise: Add another elif section to test for having only one arg present. If the one arg is nonnumeric, prompt for the seconds. If the arg is numeric, then prompt for the banner words. Use the case to determine whether it is numeric.
Intro to Shell Scripts

• Next, we look at the \texttt{until} statement. Like the \texttt{while} statement, the \texttt{until} construct causes a command list to repeatedly execute. Unlike \texttt{while}, which executes as long as conditions remain true, \texttt{until} executes as long as conditions remain \textit{false}.

• The syntax of the until is identical to the while.

Exercise: Change the loops in greeting.5 from while to until. You will have to reverse the values of the loop control variable “loop” so that the condition starts out false and ends when it becomes true. Save this as greeting.6. Run it with $ \texttt{sh} \ -x \ greeting.6 to test it.(uses the debugger option \texttt{–x})
The last of the shell looping constructs is the **for** statement. Like **while** and **until**, it causes a command list to repeatedly execute. Unlike those statements which use test exit codes to determine whether a loop executes again, **for** executes by assigning its associated loop control variable consecutive value taken from a list of string values available.

The strings can be constants or the values of variables, such as `*`. 
Intro to Shell Scripts

– The basic `for` syntax is:
  
  ```
  for var in string1 string2 string3 ...
  do
    command-list
  done
  ```

  where `var` is any user-defined variable. Do not precede it with a dollar sign

– Each `string` is a character string which will be assigned to `var`. Strings have these properties:
  
  • They are separated by a field separator character (usually a space or tab).
  • A `string` representing a filename can use shell wildcards.
  • At least one `string` must follow the keyword “in”; any number may follow.
  • Use a backslash (`\`) for line continuation if all `strings` do not fit on one line.
Intro to Shell Scripts

- Execution of a `for` statement proceeds as follows:
  - `var` is assigned the value of `string1` as if this statement is executed: `var=string1`
  - The `command-list` then executes, up to the `done` statement.
  - `var` is then assigned the value of `string2` and the `command-list` executes again.
  - The `command-list` executes once for each `string` provided. During each iteration, `var` takes on each value in turn (and may be used within the loop body as well).

Exercise: Copy the scripts for1.sh and for2.sh.
Use `cat` to look at each one. Then run for1.sh with:
$ sh for1.sh.
Then run for2.sh with:
$ sh for2.sh trains planes autos
• The for loop is very useful when you have a list of items that all need to be processed through a particular set of command. To better illustrate its use, look at the script mailsndr.sh:

```bash
for i in `cat $1`
  do
    mailx -s $3 $i < $2
    echo $i has been sent the mail $2
  done
```

• This script is designed to email a list of users the same document. The list of users must be in a file passed as `$1`. The document file is passed as `$2`, and the subject of the email is `$3`. 
Intro to Shell Scripts

Exercise: Copy the file users. Create an email message in file and save it. Now run and watch the script to send your message to everyone in the users file with:

$ sh –x mailsndr.sh users mesgfile “Your subject”

Exercise: Copy the file chngcase.sh. Cat the file and analyze its commands. Now demonstrate its use by creating a directory that contains files with all uppercase names. Move the script into that directory and run it.
Commands/Concepts

- You now have enough information to do lab 9
- Copy the file unxlab09 from the cs40 directory to your home directory.
- Have fun with Unix!