

# Introduction to Shell Scripting

## Lecture

1. Shell scripts are small programs. They let you automate multi-step processes, and give you the capability to use decision-making logic and repetitive loops.
2. Most UNIX scripts are written in some variant of the Bourne shell. Older scripts may use `/bin/sh` (the 'classic' Bourne shell). We use **bash** here.
3. Consider this sample script (you've seen it before in the **PATH** example). It can be found at `-unixinst/bin/progress`.

```
#!/bin/bash
#
# Sample shell script for Introduction to UNIX class.
# Jason R. Banfelder.
# Displays basic system information and UNIX
# students' disk usage.
#
# Show basic system information
echo `hostname`: `w | head -1`
echo `who | cut -d" " -f1 | sort | uniq | \
  egrep "^unixst" | wc -l` students are logged in.
#
# Generate a disk usage report.
echo "-----"
echo "Disk Usage Report"
echo "-----"
cd /home
# Loop over each student's home directory...
for STUDENT_ID in `bin/ls -ld unixst*`
do
    # ...and show how much disk space is used.
    du -skL /home/$STUDENT_ID
done
```

- a. All scripts should begin with a 'shebang' (`#!`) to give the name of the shell.
  - b. Comments begin with a hash.
  - c. You can use all of the UNIX commands you know in your scripts.
  - d. The results of commands in backwards quotes (upper-left of your keyboard) are substituted into commands.
  - e. Use a `$` before variable names to use the value of that variable.
  - f. Note the **for** loop construction. See the **Introduction to UNIX** manual for details on the syntax.
4. Scripts have to be executed, so you need to **chmod** the script file. Use `ls -l` to see the file modes.

## **Exercise**

1. Write a script to print out the quotations from each directory.
  - a. Did you write the script from scratch, or copy and modify the example above?
2. Create a subdirectory called **bin** in your home directory, if you don't already have one. Move your script there.
3. Permanently add your **bin** directory to your **PATH**.
  - a. It is a UNIX tradition to put your useful scripts and programs into a directory named **bin**.
4. Save your script's output to a file, and e-mail the file to yourself.
  - a. **mail myself@my.college.edu < AllMyQuotations.txt**
  - b. We hope you enjoy this list of quotations as a souvenir of this class.

## More Scripting Techniques

### Lecture

1. As you write scripts, you will find you want to check for certain conditions before you do things. For example, in the script from the previous exercise, you don't want to print out the contents of a file unless you have permission to read it. Checking this will prevent warning messages from being generated by your scripts.

- a. The following script fragment checks the readability of a file. Note that this is a script fragment, not a complete script. It won't work by itself (why not?), but you should be able to incorporate the idea into your own scripts.

```
if [ -r $STUDENT_ID/quotation ]; then
    echo
    cat $STUDENT_ID/quotation
fi
```

- b. Note the use of the **if...fi** construct. See the **Introduction to UNIX** manual that accompanies this class for more details, including using **else** blocks.
    - i. In particular, note that you **must have spaces** next to the brackets in the test expression.
  - c. Note how the **then** command is combined on the same line as the **if** statement by using the **;** operator.
  - d. You can learn about many other testing options (like **-r**) by reading the results of the **man test** command.
2. The **read** command is useful for reading input (either from a file or from an interactive user at the terminal) and assigning the results to variables.

```
#!/bin/bash
#
# A simple start at psychiatry.
# (author to remain nameless)
echo "Hello there."
echo "What is your name?"
read PATIENT_NAME
echo "Please have a seat, ${PATIENT_NAME}."
echo "What is troubling you?"
read PATIENT_PROBLEM
echo -n "HMMMMMM....  '"
echo -n $PATIENT_PROBLEM
echo "'    That is interesting... Tell me more..."
```

- a. Note how the variable name is in braces. Use braces when the end of the variable name may be ambiguous.

3. The **read** command can also be used in a loop to read one line at a time from a file.

```
while read line; do
    echo $line
    <your script code here>
done < input.txt
```

- a. You can also use `[]` tests as the condition for the loop to continue or terminate in **while** commands.
  - b. Also see the **until** command for a similar loop construct,
4. You can also use arguments from the command line as variables.

```
while read line; do
    echo $line
    <more script code here>
done < $1
```

- a. **\$1** is the first argument after the command, **\$2** is the second, etc.

### **Exercise**

1. Modify your quotation printing script to test the readability of files before trying to print them.

# Scripting Expressions

## Lecture

1. You can do basic integer arithmetic in your scripts.
  - a. `total=$(( $1 + $2 + $3 + $4 + $5 ))` will add the first five numerical arguments to the script you are running, and assign them to the variable named `total`.
  - b. Try typing `echo $(( 5 + 9 ))` at the command line.
  - c. What happens if one of the arguments is not a number?
  - d. You can use parentheses in the usual manner for grouping within **bash** math expressions.
    - i. `gccontentMin=$(( ( $gcMin * 100 ) / $total ))`

## Exercise

1. When we learned about `csplit`, we saw that we had to know how many sequences were in a `.fasta` file to properly construct the command.
  - a. Write a script to do this work for you.

```
#!/bin/bash
#
# Intelligently split a fasta file containing
# multiple sequences into multiple files each
# containing one sequence.
#
seqcount=`egrep -c '^>' $1`
echo "$seqcount sequences found."
if [ $seqcount -le 1 ]; then
    echo "No split needed."
    exit
elif [ $seqcount -eq 2 ]; then
    csplit -skf seq $1 '%^>%' '/^>/'
else
    repcount=$(( $seqcount - 2 ))
    csplit -skf seq $1 '%^>%' '/^>/' \${${repcount}}\}
fi
```

- b. Expand this script to rename each of the resultant files to reflect the sequence's *GenBank ID*.  
`>gi|37811772|gb|AAQ93082.1| taste receptor T2R5 [Mus musculus]`  
 This is shown underlined and in italics in the example above.
- i. How would you handle fasta headers without a GenBank ID?
- c. Expand your script to sort the sequence files into two directories, one for nucleotide sequences (which contain primarily A, T, C, G), and one for amino acid sequences.
- i. How would you handle situations where the directories do/don't already exist?
  - ii. How would you handle situations where the directory name already exists as a file?
  - iii. When does all this checking end???

2. What does this script do? (hint: `man expr`)

```
#!/bin/bash
gcounter=0
ccounter=0
tcounter=0
acounter=0
ocounter=0
while read line ; do
  isFirstLine=`echo "$line" | egrep -c '^>'`
  if [ $isFirstLine -ne 1 ]; then
    lineLength=`echo "$line" | wc -c`
    until [ $lineLength -eq 1 ]; do
      base=`expr substr "$line" 1 1`
      case $base in
        "a"|"A")
          acounter=`expr $acounter + 1`
          ;;
        "c"|"C")
          ccounter=`expr $ccounter + 1`
          ;;
        "g"|"G")
          gcounter=`expr $gcounter + 1`
          ;;
        "t"|"T")
          tcounter=`expr $tcounter + 1`
          ;;
        *)
          ocounter=`expr $ocounter + 1`
          ;;
      esac
      line=`echo "$line" | sed 's/^.//'`
      lineLength=`echo "$line" | wc -c`
    done
  fi
done < $1
echo $gcounter $ccounter $tcounter $acounter $ocounter
```

3. Write a script to report the fraction of GC content in a given sequence.
- a. How can you use the output of the above script to help you in this?