CSE 302LR: Introduction to Experiential Learning/Research (Section B) - Fall 2025

Lecture 3: Shell scripting and data processing

9/9/2025

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Clemen 217, 3:30 pm - 4:50 pm. In-person attendance required.

Find more on course website:
https://cse.buffalo.edu/adblab/elrr/

Shell scripting

- Taking bash as an example
  - Comments start with a `#`
  - Shebang (first line that starts with `#!`)
    - special comment that tells OS how to execute the script
  - Let's say we have a file `hello.sh` as follows:

```
#!/bin/bash
echo "Hello, world!" # prints Hello, world!
```

OS reads the first line and runs `/bin/bash hello.sh`

echo and cat

- echo: Prints a message as is

```
echo 123
echo yes
echo “Hello, world!”
```

- cat: prints a file on screen

```
cat /etc/passwd
```

Variables

- Syntax: `variable=value`
  - No space allowed.
  - Variable names: letters/digits/underscore and cannot start with digits

```
N=100
echo $N
echo ${N}
echo ${N}123
echo $N123
```

Quotes

- Single quote vs double quote
  - Single quote disables variable expansion; double quote enables variable expansion

```
N=100
echo ‘$N’  # $N
echo “$N” # 100
echo “${N}123” # 100123
```

- Backtick `` and $()
  - executes a command and expands to its output (stdout)

```
echo “Current time is `date –Iseconds`”
CWD=“$(pwd)”
echo “Current working directory is $CWD”
```

Environment variables

- For passing information to child processes

  Let's say we have two scripts and we execute a.sh

```
# a.sh
export N=100
bash b.sh

# b.sh
echo $N # 100
```

Regular expression

- `.` – “any single character” except newline
- `*` – zero or more of the preceding match
- `+` – one or more of the preceding match
- `[abc]` - any one character of a, b, and c
  - `[a-qB-Z]` – any one character within lower case a to q or upper case B to Z
  - `[^0-9]` – any one character that is not 0 to 9
- `(RX1|RX2)` either something that matches RX1 or RX2
- `^` – the start of the line
- `$` – the end of the line

grep and sed

- grep searches matching lines with a regular expression

```
grep “your-username” /etc/passwd
grep “^export PATH=” ~/.bashrc
```

- sed can be used for substituting regular expression matches

```
sed ‘s/^HISTSIZE=\([0-9]*\)/HISTSIZE=3000/’ ~/.bashrc
sed “s/\/home\/$\{USER\}/\${HOME}/” ~/.bashrc # one per line
sed “s/\/home\/$\{USER\}/\${HOME}/g” ~/.bashrc # all per line
```

Redirecting/pipe input/output

```
echo 123 >out.txt # redirect stdout to file
cat - <out.txt # redirect stdin to file

# pipe stdout to next command as stdin
pwd | sed "s/${USER}/anonuser/g"

# redirect stderr to file
sed "s/$USER" ~/.bashrc 2>err.log

# redirect both stdout and stderr to the same file
sed "s/$USER" ~/.bashrc >&out.txt
# or
sed "s/$USER" ~/.bashrc >out.txt 2>&1
# However, the following only redirects stdout to out.txt!
sed "s/$USER" ~/.bashrc 2>&1 >out.txt
```

Conditional statement

```
N=100
if [ $N –eq 100 ]; then
    echo “N is 100”
else
    echo “N is not 100”
fi
```

Loop

```
N=0
while [ N -lt 5 ]; do
      echo $N
      N=`expr $N + 1`
done
```

```
for N in 0 1 2 3 4; do
      echo $N
done
```

```
for ((N=0; N<5; ++N)); do
      echo $N
done
```

Useful reading

- Welcome to The Grymoire!

Exercise

Hints: multi-line patterns with sed
get help from "man head", "man tail" and "help read"

1. Write a script to extract all SQL statements and associated query plan dumps:
   - Usage: `extract_sql_plans.sh <input> <outputdir>`
   - `<input>` is a PostgreSQL log file
     - logs are available at https://cse.buffalo.edu/adblab/elrr/fa25/pglogs
     - Check sample input and output for 2024-09-15_115813
   - `<outputdir>` is a path to a directory that may or may not exist
     - If the directory does not exist, you need to attempt to create it.
     - If the directory exists, you do not need to do anything.
     - You may assume outputdir is never an existent file that is not a directory.
     - Extracted SQL files, in the order they appear in the log file, should be written to `<outputdir>/stmt1.sql, <outputdir>/stmt2.sql, …`
     - The associated plan, in the order they appear in the log file, should be written to `<outputdir>/stmt1.plan, <outputdir>/stmt2.plan, …`
     - No plan file should be written for a SQL statement with execution error.
   - When the number of arguments is smaller than 2
     - print the usage line to stderr
   - The script may only print a single line to stdout containing a single number indicating how many SQL statements were found in the log file.
   - Any other error messages, including those from tools, may only be printed to stderr.

Demo your script at 4:30 pm.