Name: Justin Weigle

UNIX Shells

1. What is a "shell" ?

   A "command interpreter" as the textbook puts it.
   In other words, it is a command line interface that contains the code to execute certain commands provided to it by the user.

2. Determine the "official" name for each of these

   - sh **shell**
   - jsh **javascript system shell**
   - ksh **korn shell**
   - csh **C shell**
   - bash **Bourne again shell**
   - tcsh **TENEX C shell**
   - ash **Almquist shell**

3. What shell are you currently using?

   zsh (Z shell)
   I knew this because I picked it, but it can be identified using "echo $SHELL"

4. How do you change your default shell?

   chsh
   It is also recommended to run "which {desired_shell_name}" prior to running chsh as it will tell you the full path to the shell you want, which chsh expects.

5. How do you change your currently running shell (not permanently)?

   {desired_shell_name}
   So for example, I'm running zsh, if I wanted to run bash temporarily I would type: bash

UNIX Plumbing

The plumbing operators allow you to redirect output to/from a file or to another program:

- command > file - run command, send its *standard output* to file
- command < file – run command, read *file* as its *standard input*
- command 2> file – run command, send its *standard error* to file
- command1 | command2 – run command 1 and 2, send command 1's standard output to command 2's standard input
- command 2>&1 – run command, merge its standard error to its standard output

1. Use the *find* command to list all of the files in a directory

```
find {directory_name}
```

2. Use the find command as before, but pipe the output to one of the *pagination* programs (*more* or *less*) to view the output one page at a time:

```
find {directory_name} | less
```

3. Show the output of the *ps* command, but only look for lines that contain bash

```
ps | grep “bash”
```

4. Use the *git* command line utility to list all of the available branches that include the string “release-to-test”

```
git branch | grep “release-to-test”
```

5. The command to find lines of a file that contain the text “monty burns” (ignoring case), but then only show the 3rd, 4th, and 8h column of the text using “~” as a delimiter

```
cut -s -f 3,4,8 -d “~” {file} | grep -i “monty.burns”
```

Programming with *sed*

Find one of the millions of references on the *sed* tool then answer these questions (some of these are basically Hacker Rank questions, so theres some easy Hackos here). Remember, *sed* expects its input to come from standard input, and it writes its output to standard output – so you'll want to use the pipe operations:

6. Replace the text "Atlanta Falcons win" with "New England Patriots win", ignoring case and all occurrences on a line

   sed -i "s/Atlanta Falcons win/New England Patriots win/gI" {file}

7. Replace the text "Atlanta Falcons win" with "New England Patriots win", ignoring case and only occurrences as the start of a line.

   sed -i "s/Atlanta Falcons win/New England Patriots win/I" {file}

8. Use *character types* to match two or more commas in a row and replace them with one comma

   sed -i "s/,\{2,\}/,/g" {file}

9. Use *regular expression grouping* to match a ship.edu email address (e.g. aA1234@ship.edu) and re-write it to be a "cs.ship.edu" email address, but in all lower case: aa1234@cs.ship.edu.

   sed -i "s/\(\w\{6\}\)@ship.edu/\L\1@cs.ship.edu/g" {file}

   or to match only ccNNNN@ship.edu where c = character and N = number:

   sed -i "s/\([a-zA-Z]\{2\}[0-9]\{4\}\)@ship.edu/\L\1@cs.ship.edu/g" {file}

Programming with *awk*

Hacker Rank also has a section *awk*, which may be useful and vaguely similar:

10. Write an *awk* script that can verify when a line contains exactly 4 fields, separated by spaces (see HR: Awk #1)

```
#!/usr/bin/awk -f
{ if (/^\S+ \S+ \S+ \S+$/) print “True: “ $0 ; else print “False” }
```

11. Write an *awk* script that will compute the average of three numbers (fields 2, 3, 4) and print the first field, a colon, and the average on a line (see HR: Awk #2)

```
#!/usr/bin/awk -f
{ print $1 " : " (($2 + $3 + $4) / 3) }
```

UNIX Environment Variables

Lookup the following environment variables, describe what they do and when you would need to change them:

12. PATH **Tells the shell which directories to search for executables in. May need changed if the shell can’t find a specific executable you want access to.**
13. TERM **Sets the default terminal emulator. Change if you want a different default.**
14. LD_LIBRARY_PATH **Path used for debugging a new or non-standard library. Change if you need to test a library that isn’t standard.**
15. DISPLAY **Defines: hostname:D.S or more recently hostname:D. Where D = display and S = screen. May need changing if there are issues with X11 forwarding**
16. HOME **Determines where the current user’s home directory is. Change if a different home directory is desired.**
17. PS1 **Format of the terminal prompt. Change to customize look/info of prompt.**
18. TEMP **Path to directory for programs to make temporary files. Change if the current path is not where you want temporary files to go.**
19. EDITOR **Default text editor executable. Change if you want to set a different default editor.**
20. PAGER **Default pager to use (such as less, and cat being a fallback). Change to use a different program as default.**
21. Write the command to set an environment variable to a new value:

export {variable}={path_or_executable_name}

To make the changes permanent, add to .profile or .zprofile or any startup file when running default shell.

22. Write the command show the value of an environment variable

    printenv {env_variable}

23. Write the command to show all environment variables
    printenv

24. Write the command to delete an environment variable
    unset {env_variable}

25. The "bash" shell program interprets the prompt specially, find a description of the special commands you can use in your prompt, and make yourself a cool prompt below. Show the "set" command, and then also show what it looks like when you use it:

```
set PS1='[\A] \u@\s \w \$ '
export $1

[00:00] user@shell ~ $
(\w is the full directory path)
```

UNIX Shell Scripting

Use "bash" to write the following as shell scripts (there are thousands of shell script refs on the internet)

26. Print all the odd numbers from 1 to 99 on the terminal

```
#!/bin/sh

i=0
while (( $i < 100 ))
do
   if (( $i % 2 != 0 ))
   then
      echo $i
   fi
   i=$(( i + 1 ))
done
```

27. Print a times table from 1 to 10 on each dimension

```
#!/bin/sh

i=1
while (( $i <= 10 ))
do
   j=1
   while (( $j <= 10 ))
   do
      k=$(( $i * $j ))
      if (( $k < 10 ))     # conditional printf not necessary, but looks better
      then
         printf $(( $k ))"  "       # with 2 spaces here
      else
         printf $(( $k ))" "
      fi
      j=$(( j + 1 ))
   done
   i=$(( i + 1 ))
   printf "\n"
done
```

----------------------------------- without conditional printf ---
----

```
#!/bin/sh

i=1
while (( $i <= 10 ))
do
   j=1
   while (( $j <= 10 ))
   do
      printf $(( i * j ))" "
      j=$(( j + 1 ))
   done
   i=$(( i + 1))
   printf "\n"
done
```

28. Write a shell *function* that will take two numbers and determine if a is a multiple of b, then use that function to write another function to determine if a number *a* is prime, and then use a for loop to display prime numbers between 2 and 31.

```
#!/bin/sh
is_mult_of () {
   if (( $1 % $2 == 0 ))
   then
      return 0 #0 = true
   else
      return 1 #1 = false
   fi
}
is_prime () {
   if is_mult_of $1 $2
   then
      return 1 #1 = false
   else
      return 0 #0 = true
   fi
}

for i in {2..31}
do
   if (( $i == 2 ))
   then
      echo $i
      continue
   fi
   j=$(( i - 1 ))
   while (( j >= 2 ))
   do
      if ! is_prime $i $j
      then
         break
      fi
      if (( $j == 2 ))
      then
         echo $i
      fi
      j=$(( j - 1 ))
   done
done
```

29. Loop over all of the files in a directory and if the file contains the text “bob”, add it to a shell variable. After processing all files, display the final list of files, all on one line. The shell script should produce no other output.

```
#!/bin/sh

filelist=()
for i in *
do
   if ( grep "bob" $i > /dev/null )
   then
      filelist+=$i
   fi
done
echo ${filelist[@]}
```

30. Consult a manual on the "test" command, and then write a bash shell function that will loop over all files in a directory whose named ends in ".c". For each C file, use the test utility to see if there is a matching ".o" file, if there is, look to see if the .c file is newer than the .o. If the .o doesn't exist or the .c file is newer, then compile the .c file into the .o file. If the compilation breaks, abort the shell script. Finally, look to see if all the .o files are newer than the name of an executable given on the command line. If the executable is not present, is not executable, or is older than any of the .o files, recompile the executable using all of the .o files in the directory. *Hint: this one might go better if you use shell variables.*

```
#!/bin/sh

set -e

for i in *.c
do
   j=${i::-2}
   if ( test -e $j.o )
   then
      if ( test $i -nt $j.o )
      then
         gcc -c $i
      fi
   else
      gcc -c $i
   fi
done

objfiles=*.o
echo "Please input executable name"
read execname

if ( test -e $execname )
then
   if ( test -f $execname )
   then
      gcc -o $execname ${objfiles[@]}
      break
   fi
   for i in ${objfiles[@]}
   do
      if ( test $execname -ot $i )
      then
         gcc -o $execname ${objfiles[@]}
         break
      fi
   done
else
   gcc -o $execname ${objfiles[@]}
fi
```

31. Perform some background research and find out the name(s) of the shell script that bash will execute when you log in, and when you log out. Modify the shell script to change your prompt to something fun (your choice), modify the path to include your own personal bin directory, and create an alias for the vi command to be edit.

.bash_profile:

```
PS1='something fun'
PATH=$PATH:~/bin
alias edit='vi'
```

UNIX Job Control

32. Show how to run a command in the background:

    Some programs allow running in the background by flag or default, but the universal option is:
    {command} &

33. Describe how to kill a command that is currently running:

    Identify the PID by: ps -e
    kill {PID} (default sends SIGTERM, use -KILL flag to send kill)
    Or if you're running it in the cli you're in, you can ctrl-c

34. Describe how to suspend a command that is currently running:

    Use ctrl-z if the program is running in your current cli.
    Otherwise you can use kill -TSTP {PID}

35. Describe how to put a suspended command into the background:

    ctrl-z followed by bg

36. Describe how to restore a suspended command to the foreground:

    fg

37. Describe how to run a command and have it continue to run after you logout:

    nohup {command} &