Evaluation: 28 Questions + 1 Script **Name: _________________________**

Important Instructions

1. Read all instructions and both sides of both pages.
2. Manage your time when answering questions on this test.
   Answer the questions you know, first.
3. The Multiple Choice section is worth 8% of the 25%
   There are no penalties for wrong answers.
4. The Script Writing section is worth 17% of the 25%

Multiple Choice Section - 28 Questions - 8% of 25%

(Office use only: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28)

1. If **`foo`** is a script containing the line **`TERM=vt100 ; export TERM`**, what is the output of the following sequence of **`bash`** commands:
   `TERM=linux ; ./foo ; echo $TERM`
   † **a**. **`linux`**
   b. **`vt100`**
   c. **`foo`**
   d. **`TERM`**
   e. **`$TERM`**

2. If **`x=8`** and **`y=9`** then which of the following **`bash`** command lines outputs only the word **`foobar`** (and nothing else)?
   † **a**. **`[ x = x ] && echo foobar`**
   b. **`[ x -ne y ] && echo foobar`**
   c. **`[!x = y] && echo foobar`**
   d. **`[x -ne y] || echo foobar`**
   e. **`[x!=x] || echo foobar`**

3. What is the output of the following sequence of **`bash`** commands:
   `echo wc >wc ; wc wc >wc ; cat wc`
   † **a**. **`0 0 0 wc`**
   b. **`1 1 3 wc`**
   c. **`1 1 2 wc`**
   d. no output
   e. **`wc`**

4. If **`guru=linus`** then which one of the following **`case`** patterns will match this statement: **`case "$guru" in`**
   † **a**. **`*) echo yes ;;`**
   b. **`lin?) echo yes ;;`**
   c. **`"linu?") echo yes ;;`**
   d. **`(*nus  echo yes ;;`**
   e. **`[linus] | [LINUS]) echo yes ;;`**

5. Which **`bash`** command sequence correctly searches for the string **`foobar`** and then prints **`YES`** if it is found inside the group file?
   † **a**. **`if grep foobar /etc/group ; then echo YES ; fi`**
   b. **`if [ grep foobar /etc/group ] ; then echo YES ; fi`**
   c. **`if test foobar /etc/group ; then echo YES ; fi`**
   d. **`if test foobar = /etc/group ; then echo YES ; fi`**
   e. **`if [ test foobar /etc/group ] ; then echo YES ; fi`**

6. What is the value of variable **`foobar`** at the end of the loop that starts:
   `for foobar in 5 5 $# $? 6 1 ; do`
   † **a**. **`1`**
   b. **`5`**
   c. **`6`**
   d. **`0`**
   e. the value is undefined

7. A shell script named **`bar`** is executed as follows:
   `./bar a  "b c"  ’a  ’`
   Inside the script is the line: **`head $@`**
   How many arguments are passed to the **`head`** command inside the script?
   † **a**. **`4`**
   b. **`5`**
   c. **`6`**
   d. **`3`**
   e. **`2`**

8. If a script named **`bar`** contains a loop that starts:
   `for i in $* ; do`
   and the script is executed using this command line:
   `./bar a ’ b d ’ e f " g h " a`
   how many times will the loop iterate?
   † **a**. 8 iterations
   b. 9 iterations
   c. 7 iterations
   d. 6 iterations
   e. 12 iterations

9. If variable **`bar`** might contain nothing (a null value - defined but empty), which **`bash`** command sequence correctly tests for this and prints **`YO`**?
   † **a**. **`if [ "$bar" = "" ] ; then echo YO ; fi`**
   b. **`if [ $bar -eq : ] ; then echo YO ; fi`**
   c. **`if [ $bar -eq "" ] ; then echo YO ; fi`**
   d. **`if [ ’’$bar’’ = ’’’’ ] ; then echo YO ; fi`**
   e. **`if [ "$bar" = * ] ; then echo YO ; fi`**

10. What is the output of the following sequence of **bash** commands:
    `x=0 ; y=1 ; test ! -z $x ; echo $?`
  - † **a**. `0`
  - b. `1`
  - c. the number 1 or 0 followed by another 1 or 0 on a new line
  - d. `test: $x: integer expression expected`
  - e. no output

11. What is the output of the following sequence of **bash** commands:
    `echo 'Good-day World' | tr -d 'W'`
  - † **a**. `Good-day orld`
  - b. `GoodWay World`
  - c. `GoodWay orld`
  - d. `GoodW ay World`
  - e. `Good-day World`

12. What is the output of the following sequence of **bash** commands:
    `echo 'Good-day World' | sed -e 's/^/99/g'`
  - † **a**. `99Good-day World`
  - b. `99Good-day 99World`
  - c. `99ood-day World`
  - d. `99ood-day 99orld`
  - e. `Good-day World`

13. Which **sed** command turns the string **Good-Day; World-Traveller** into **Day-Good; World-Traveller**:
  - † **a**. `s/\([^-]\+\)-\([^-;]\+\)/\2-\1/`
  - b. `s/\(![-]\+\)-\(![-;]\+\)/\2-\1/`
  - c. `s!/\([-]\+\)-\([-;]\+\)/\2-\1/`
  - d. `s/\([a-z]*\)-\([a-z]*\)/$2-$1/`
  - e. `s/\(.*\)-\(.*\)/$2-$1/`

14. Which **sed** command finds every occurrence of three adjacent letters and reverses them: (e.g. **dogcatcow** would become **godtacwoc**; but, **do3ca;c.o** would not change).
  - † **a**. `s/\([a-z]\)\([a-z]\)\([a-z]\)/\3\2\1/g`
  - b. `s/\([^0-9]\)\([^0-9]\)\([^0-9]\)/\3\2\1/g`
  - c. `s/\(![0-9]\)\(![0-9]\)\(![0-9]\)/\3\2\1/g`
  - d. `s/\(![0-9]\)\{3\}/\3\2\1/g`
  - e. `s/\(^[0-9]\)\{3,3\}/\3\2\1/g`

15. Which **sed** command finds every line that ends in the digits **123** and removes the first occurrence of the string **xyzzy** from those lines:
  - † **a**. `/123$/s/xyzzy//`
  - b. `/[0-9][0-9][0-9]$/s/xyzzy//`
  - c. `s/xyzzy.*123$/123/`
  - d. `s/^.*xyzzy\(.*123\)$/\1/`
  - e. `/xyzzy/s/[0-9][0-9][0-9]\$//`

16. Which **sed** command finds every line that contains the four digits 1, 2, 3, and 4 in ascending order (but with any number of any characters in between) and deletes the first letter on these lines:
  - † **a**. `/1.*2.*3.*4/s/[a-zA-Z]//`
  - b. `/[0-9].*[0-9].*[0-9].*[0-9]/s//[a-zA-Z]/`
  - c. `/[1-1]*[1-2]*[1-3]*[1-4]/s/a-zA-Z/d`
  - d. `/(1)*(2)*(3)*(4)/[a-zA-Z]\1\2\3\4/d`
  - e. `/\1.*\2.*\3.*\4/[a-zA-Z]/d`

17. Which **sed** command deletes only lines that do not contain any lower-case vowels (including blank lines):
  - † **a**. `/[aeiou]/!d`
  - b. `/[^aeiou]/d`
  - c. `/![aeiou]/d`
  - d. `s/![aeiou]\+//g`
  - e. `s/^[!aeiou]*$//`

18. Which **sed** command deletes only lines that contain at least one non-digit?
  - † **a**. `/[^0-9]/d`
  - b. `/[0-9]/!d`
  - c. `/![0-9]/d`
  - d. `s/![0-9]\+//g`
  - e. `s/^[!0-9]*$//`

19. Which **sed** command takes every occurrence of an asterisk (*****) followed by a period (**.**) and reverses all of them:
  - † **a**. `s/\*\./.*/g`
  - b. `s/\(*\)\(.\)/\2\1/g`
  - c. `s/\(\*\)\(\.\)/\2\1/`
  - d. `s/(\*)(\.)/\2\1/g`
  - e. `s/[*][.]/\2\1/g`

20. Which **awk** command outputs non-blank lines where the line number is an exact multiple of the number of fields:
  - † **a**. `1 <= NF && 0 == (NR % NF) { print }`
  - b. `NR > 0 && (NF % NR) == 0 { print $0 }`
  - c. `1 <= RN && 0 == (FN % RN)`
  - d. `/FN >= 1/ { if (RN % FN == 0) print $0 }`
  - e. `/FN >= 1 && (RN % FN == 0)/ { print }`

21. Which **awk** command outputs all lines where the second field is larger than the line number:

† a. `NR < $2`
b. `$2 > $NR { print }`
c. `$2 > $LN { print $0 }`
d. `$2 > LN`
e. `$NF < $2 { print }`

22. Which **awk** command outputs the third field of lines that have at least three fields:

† a. `3 <= NF { print $(1+3-1) }`
b. `FN >= 3 { print $3 }`
c. `2 < $FN { print $3 }`
d. `/$3/ { print $2+1 }`
e. `/$3 != ""/ { print $1+1+1 }`

23. Which **awk** command outputs the sum of the first two fields on every line:

† a. `{ print $1+$2 }`
b. `{ print $(1+2) }`
c. `RN >= 2 { print $(1+$2) }`
d. `$FN >= 2 { print $(1+$2) }`
e. `$1 > 0 && $2 > 0 { print $(1+$2) }`

24. Which **awk** command outputs lines where the last field is numeric?

† a. `$NF ~ /^[0-9]+$/`
b. `$FN ~ /^[0-9]*$/ { print $0 }`
c. `FN ~ /[0-9]\+/ { print $0-FN }`
d. `$FN == "^[0-9]\+$" { print }`
e. `NF == /^[0-9][0-9]*$/ { print $0 }`

25. Which **awk** command outputs all lines where the second field contains only the string **xyzzy** and where the third field exactly matches the first field:

† a. `$1 == $3 && "xyzzy" == $2`
b. `$2 ~ /^xyzzy/ && $3 == $1`
c. `$2 ~ /^xyzzy/ && $3 ~ /$1/`
d. `$2 == /^xyzzy$/ && $3 == $1`
e. `$3 == $1 || "^xyzzy$" ~ $2`

26. Which **awk** command outputs all lines where the first field begins with the letters **abc** and the last field ends with the letters **xyz**:

† a. `$NF ~ /xyz$/ && $1 ~ /^abc/`
b. `$1 ~ /abc$/ && $NR ~ /^xyz/`
c. `$1 = /abc/ && $FN = /xyz/`
d. `NR = /^xyz$/ && $1 = /^abc$/`
e. `$1 == "abc" && $FN == "xyz"`

27. What is the output (if any) of this **bash** shell program fragment? (There are blanks between all the letters in the word list sections of the **for** loops.)

```
str=''
for x in   s l i c k  r u n  o x   ; do
    for y in   l i n u x  r o c k s   ; do
        if [ $x = $y ] ; then
            str=$str$y
        fi
    done
done
echo $str | sed -e 's/^\(.....\)/\1 /'
```

† a. `slick runox`
b. `linux rocks`
c. `s`
d. `x`
e. no output

28. What is the output of this **bash** shell program fragment? (There are blanks between all the letters in the word list section of the **for** loop.)

```
in='' out=''
for char in   l i n u x   ; do
    case "$char" in
    L|l)  in=$in$char ; out=${out}L ;;
    [Xx]) in=$in$char ; out=${out}X ;;
    esac
done
echo "linux rox" | tr $in $out
```

† a. `LinuX roX`
b. `linux rox`
c. `louux rox`
d. `lINUx ROx`
e. `LINUX ROX`

Script Writing - this script is marked out of 100 marks total
This script is worth 17% of the total 25% for this exam

Design and then write (in that order!) the following **`bash`** script that will extract some data from a Unix-style password file and then use the data.

For best marks, your finished script on paper must have the appearance that you **designed** it before you started coding it. Many **insertions** and/or excessive **crossed-out** material will cost you marks. Write your PDL before you code!

Each of the steps below has been written so that you can code it **independently** of previous steps, using the given information. Do as much as you can.

Lines in the Unix password file have this 7-field colon-separated record format:

```
userid:password:uid:gid:lastname firstname:home:shell
```

Follow these steps to write this script:

1. *[Marks: 20]* **Document your code.** The script you write does not need an assignment label or purpose section; but, it must follow the other course script writing guidelines, especially with regard to the interpreter line, path, file creation mask, and **block comments**. Include the associated **step number** in each of your block comments.

   For best marks, document with block comments *all* the steps, even steps that you do not know how to code. (Leave the code empty.)

   Remember to **validate all inputs**. Issue appropriate useful, helpful **error messages** if you find incorrect, missing, or unusable input.
2. *[Marks: 12]* Script syntax: **`$0 fullname [ pathname ]`**
   Input: The script will take either one or two command line arguments.

   The first argument is a person's full name (e.g. **`"Ian D. Allen"`**), and it is mandatory. If the argument is missing, generate a useful, helpful error message and exit the script with status **1.** Transfer the first argument to a variable named **fullname**.

   The second command line argument is a password file **pathname**, and it is optional. If the second argument is present, transfer it to a variable named **pathname**. If it is missing from the command line, issue a good prompt and read the **pathname** from the user.
3. *[Marks: 3]* If the user's input is blank (the **pathname** is an empty string), set the **pathname** to be the usual Unix password file named **`/etc/passwd`**.
4. *[Marks: 5]* If the **pathname** is not readable, issue a useful, helpful error message and exit the script with status **2.**
5. *[Marks: 5]* If the **pathname** is not a plain file, issue a useful, helpful error message and exit the script with status **3.**
6. *[Marks: 5]* If the **pathname** file is zero size (is empty, contains no data), issue a useful, helpful error message and exit the script with status **4.**

7. *[Marks: 3]* Convert the text string in **fullname** to all lower-case letters.
8. *[Marks: 3]* Split the text string in **fullname** on white-space, and put the first field into a variable named **firstname**.
9. *[Marks: 3]* Split text string in **fullname** again, and put the *last* field (not the second field) into a variable named **lastname**.
10. *[Marks: 3]* Remove all the single-quote characters from the text in **lastname** (e.g. change **`o'donnell`** into **`odonnell`**).
11. *[Marks: 8]* Refer to the colon-separated, 7-field record format of the Unix pasword file, above. Process the password file named by **pathname** and match the user-entered **lastname** and **firstname** together against the field containing the same information in the password file. (Get the order of the two parts of the name right when you do the match.) The match must be an exact match; you may assume that the password file has exactly one blank between the last and first names in the name field. If the two user-entered names exactly match the corresponding two names in the name field in the password file, extract just the **`userid`** field from the matching line and save the userid into a variable named **userid**. Remember: The field separator in the Unix password file is the colon character ("**`:`**").
12. *[Marks: 6]* If the above search failed to match any name, issue a good, useful, helpful, fully-detailed error message and exit the script with status **5.**
13. *[Marks: 2]* Create a variable named **foo** that contains the pathname **`/home/`***uuu***`/*foo*`** where the string *uuu* is replaced by the **userid** just found in the password file.
14. *[Marks: 2]* Create a variable named **bar** that contains the pathname **`/home/`***uuu***`/*bar*`** where the string *uuu* is replaced by the **userid** just found in the password file.
15. *[Marks: 8]* If either the pathname in variable **foo** or the pathname in variable **bar** are not readable, plain files, issue a good, useful, helpful, fully detailed error message and exit the script with status **6.**
16. *[Marks: 6]* Count the number of lines of differences between the two files whose names are in **foo** and **bar** and display the count of the lines as follows:

    ```
    File "fff" and "bbb" difference = 'nnn' lines.
    ```

    where the two strings *fff* and *bbb* are replaced by the two file names, and the string *nnn* is replaced by the line count of the differences between the files.
17. *[Marks: 2]* If the count of the number of lines of differences between above two files is less than 10 lines, also output the line:

    ```
    Almost same: < 10 lines of difference output.
    ```
18. *[Marks: 4]* Display the first 15 lines of the file whose name is in the variable **foo**, with each line preceded by its line number.

Reminder: Did you follow all the CST8129 script coding conventions?

Answer Key - CST 8129 − Ian Allen − Fall 2002 - CST 8129 Final Exam - 25%

Office use only: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

```
  1. a
  2. a
  3. a
  4. a
    5. a
    6. a
    7. a
    8. a
  9. a
 10. a
 11. a
 12. a
   13. a
   14. a
   15. a
   16. a
 17. a
 18. a
 19. a
 20. a
   21. a
   22. a
   23. a
   24. a
 25. a
 26. a
 27. a
 28. a

Count of a:  28  100%

With 5 choices: 28
  1 2 3 4 5 6 7 8 9 10 11
  12 13 14 15 16 17 18 19
  20 21 22 23 24 25 26 27
  28

Macro .cmd splits: 9
Macro .ans splits: 0
```