**Regis University CC&IS**

**CS465 Unix**

**Lab Homework #6**

**Matt Hartigan**

**27-Apr-2019**

**\*answers below in red\***

Complete the following 7 exercises on a Unix system. Fill in your answers on this document.

Points that will be given for each correct answer are listed by each question.

NOTE: You will have 1 ½ weeks to complete it. You should begin this homework in week 6, as soon as you have turned in homework 5.

See the end of the file for submission instructions.

1. The **sort** command (*13 points total, allocation noted below*)

First create a file called **homesforsale**, such that each line lists a **six-digit MLS number**, a **home-type** (include condo, townhome, duplex, singlefam), a number of **square feet**, a number of **bedrooms**, a number of **bathrooms**, and a **price** in dollars.

Example Lines:  
444555 condo 930 2 1 99999

123456 duplex 1200 3 1.5 160900

333666 duplex 1333 3 2 185555

222999 townhome 850 2 2 83333

345678 singlefam 2300 4 2.5 250000

Your file should contain a minimum of **20** lines, and include several homes in each of the four home-type categories.

1. Submit a listing of the file you created (*1 point*).

000000 condo 1000 2 1 100000

111111 duplex 1500 4 2 150000

222222 duplex 1200 3 2 125000

333333 townhome 1333 3 2 200000

444444 singlefam 2000 3 3 185000

555555 singlefam 2500 4 3 199000

666666 duplex 1000 2 2 80000

777777 condo 990 1 1 90000

888888 condo 995 2 1 95000

999999 townhome 667 1 1 70000

000001 singlefam 1250 3 2 155000

111112 condo 1300 2 1.5 160000

222223 singlefam 3000 6 5 210000

333334 duplex 1260 3 2.5 145000

444445 condo 1310 2 1 140000

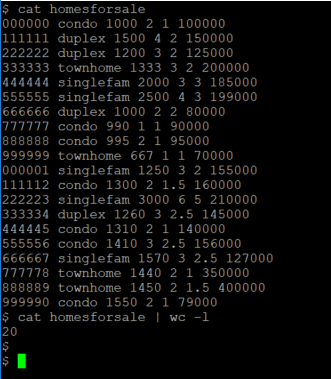
555556 condo 1410 3 2.5 156000

666667 singlefam 1570 3 2.5 127000

777778 townhome 1440 2 1 350000

888889 townhome 1450 2 1.5 400000

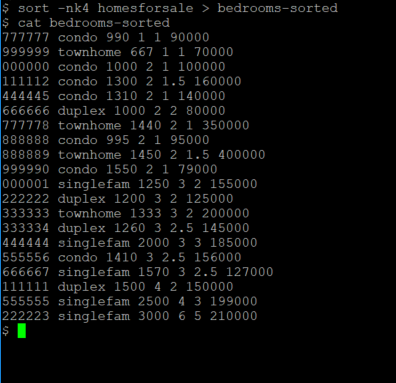
999990 condo 1550 2 1 79000



Then submit the commands and scripts that will do the following:

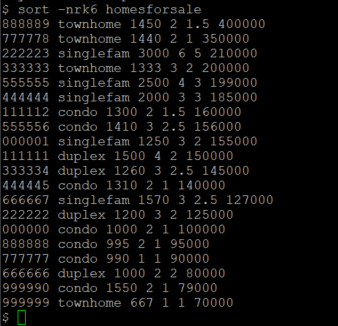
1. Write a command that will sort the file by **number of bedrooms** and store the resulting output in a file called **bedrooms-sorted** (*1.5 points*).

sort -nk4 homesforsale > bedrooms-sorted



1. Write a command that will sort the file by **price** in reverse order (high to low) and display the results (1.5 *points*).

sort -nrk6 homesforsale



d) Create a script file called **choosesort**, which will ask the user what to sort by.

The user may choose to sort by:

price alone or

square footage alone or

home-type and price or

(i.e. by type first, then by price within each home-type).

home-type and square footage or

(i.e. by type first, then by square footage within each home-type).

The script should display the chosen sorted list to the screen (9 *points*).  
 Place script code here:

#!/bin/ksh

echo "Select option you would like to sort by:"

select choice in "Price alone" "Square footage alone" "Home type and price" "Home type and square footage" "Exit"

do

case $choice in

"Price alone")

sort -nk6 homesforsale;;

"Square footage alone")

sort -nk3 homesforsale;;

"Home type and price")

sort -k2,2 -k6n,6 homesforsale;;

"Home type and square footage")

sort -k2,2 -k3n,3 homesforsale;;

"Exit")

echo "Exiting script now."

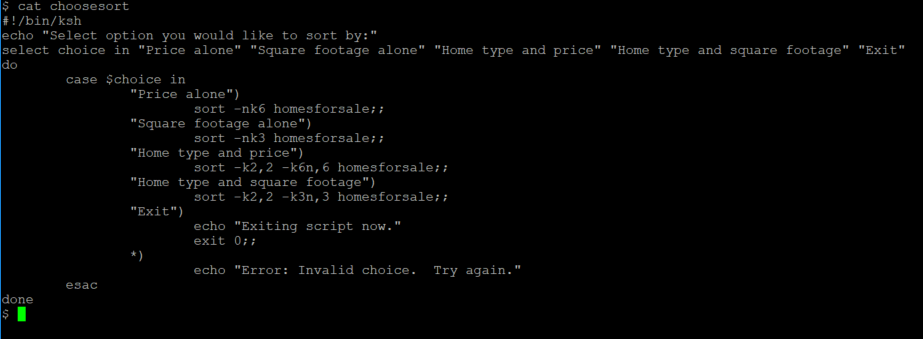
exit 0;;

\*)

echo "Error: Invalid choice. Try again."

esac

done



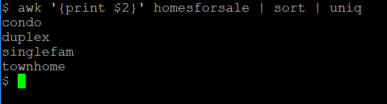
2. The **awk** command (*21 points total, allocation noted below*)

Give the **awk** commands and/or scripts containing **awk** commands that will do the following. Use the same **homesforsale** data file that you created for exercise #1.

***Notes:*** i) Do not use any **grep** commands in solving these problems.

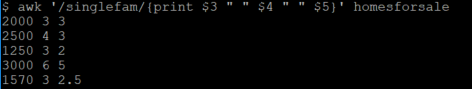
ii) If you create an **awk** program file, you must include ***both*** the   
 **awk program file listing** and the **awk command** used to run it.

a) Display a list of all **home-types** listed in the file, with duplicate listings eliminated. (*2 points*)

awk '{print $2}' homesforsale | sort | uniq  
  


b) Print out the **square footage, number of bedrooms and number of bathrooms** of all **single family homes**. (2 *points*)

awk '/singlefam/{print $3 " " $4 " " $5}' homesforsale



c) Write a script containing some **awk** command(s) which will:

- Print out just the **price** of each **condo in the file** (*2 points*)

- Then have **awk** find and print the **MLS number** **and price** of the **condo** with the **lowest price**, identified as such. (You may assume there is only one lowest price – no ties). (*4 points*)

awk program file: awkprog2c:

BEGIN {print "List of available condo prices:"

minprice=0

mls=0

}

/condo/ {

print "$" $6

if ($6 < minprice || minprice == 0)

minprice=$6

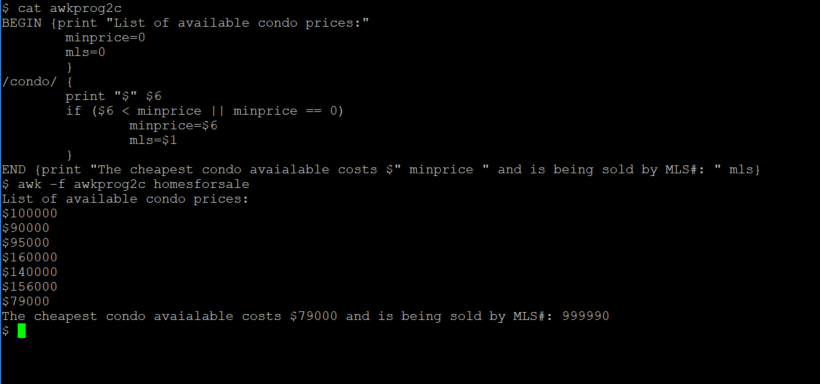
mls=$1

}

END {print "The cheapest condo avaialable costs $" minprice " and is being sold by MLS#: " mls}

awk command used to run:

awk -f awkprog2c homesforsale



1. Print a list of **MLS number** and **square footage** for all **duplexes** with square footages higher than **1200**, sorted by square footage from highest TO lowest. (*5 points*)

awk program file: awkprog2d:

BEGIN {print "List of all duplexes with square footage >1200:"}

/duplex/ {

if ($3 > 1200)

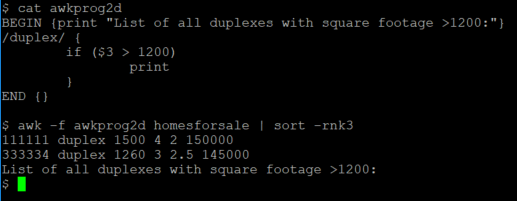
print

}

END {}

awk command used to run:

awk -f awkprog2d homesforsale | sort -rnk3



1. Write an **awk** program and **awk** command to print the **average** price of all the **townhomes** in this file, identified as such. (*6 points*)

awk program file: awkprog2e:

BEGIN {print "Average price of all townhomes in the input file:"

numtownhomes=0

sum=0

}

/townhome/ {

numtownhomes++

sum+=$6

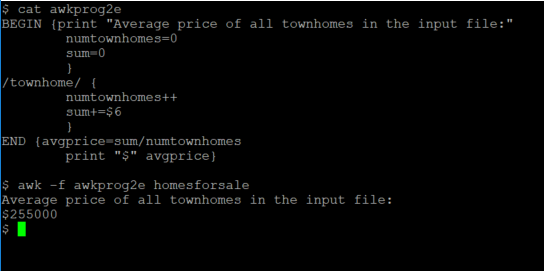
}

END {avgprice=sum/numtownhomes

print "$" avgprice}

awk command used to run:

awk -f awkprog2e homesforsale



3. (*4 points*) The **/etc/group** file is formatted as follows:

**group-name:password:groupid:group-list-usernames-comma-delimited**

Using the **/etc/group** file on our system, determine which groups on the system actually contain specific usernames (i.e. non-empty groups).   
  
For those groups, display the **group id** along with their **group name** sorted by group id, highest to lowest.

awk program file: awkprog3:

{

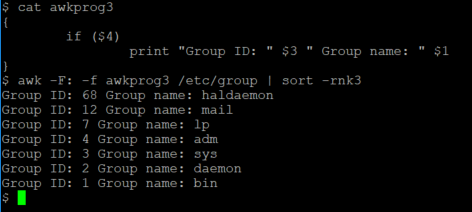
if ($4)

print "Group ID: " $3 " Group name: " $1

}

awk command used to run:

awk -F: -f awkprog3 /etc/group | sort -rnk3



4. Program Development on Unix (*17 points total, allocation noted below*)

Create two very simple C source code files: hello.c and bye.c

- In **hello.c**, function **hello** should print out "Hello World!"

- In **bye.c**, function **bye** should print out "Goodbye Everyone!"

(Note: These functions should be declared as **void** functions to prevent compiler errors.)

Also create header files **hello.h** and **bye.h**, containing the prototypes for **hello.c** and **bye.c**.

Next write a **main.c** that will call above 2 functions so that the result will display the following:

Hello World!

Goodbye Everyone!

## (Note: **main** should be declared as an **int** function to prevent compiler errors.)

## There should NOT be any **cout** or **printf** statements (or any other output statements) in **main.c**. All output should be done by the functions **hello** and **bye**

## a) Turn in a listing of all the **.c** and **.h** files you created:

## hello.c (*2 points*)

#include <stdio.h>

#include "hello.h"

/\*\*\*\*\*\*\*\*\*\* hello function \*\*\*\*\*\*\*\*\*\*/

void hello ()

{

printf("Hello World!\n");

}

## hello.h (*1 point*)

void hello(); /\* function prototype for hello \*/

## bye.c (*2 points*)

#include <stdio.h>

#include "bye.h"

/\*\*\*\*\*\*\*\*\*\* bye function \*\*\*\*\*\*\*\*\*\*/

void bye ()

{

printf("Goodbye Everyone!\n");

}

## bye.h (*1 point*)

void bye(); /\* function prototype for bye \*/

## main.c. (*2 points*)

#include <stdio.h>

#include "hello.h"

#include "bye.h"

int main(void);

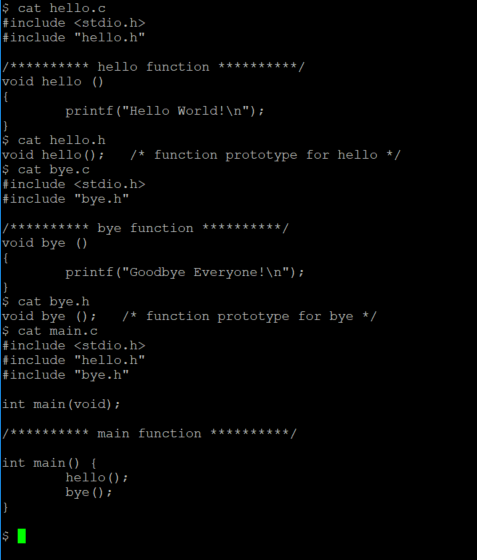
/\*\*\*\*\*\*\*\*\*\* main function \*\*\*\*\*\*\*\*\*\*/

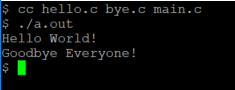
int main() {

hello();

bye();

}





## Then do the following:

Create a make file called **makegreet** which will ***separately*** compile all of the files using **cc** or **gcc** (whichever is available on your system) and then create an executable called **greet**. Include all dependencies.

## b) Turn in a listing of your make file, and its filename. (*6 points*) listing of makegreet:

## greet: main.o hello.o bye.o

## cc main.o hello.o bye.o -o greet

## main.o: main.c hello.h bye.h

## cc -c main.c

## hello.o:

## cc -c hello.c hello.h

## bye.o:

## cc -c bye.c bye.h

## Use the **makegreet** makefile to create the **greet** executable.

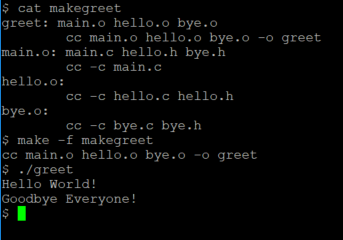
## c) What command did you use? (*2 points*)

## make -f makegreet

## Run the executable.

## d) What command did you use? (*1 point*)

## ./greet



5. Program Development on Unix (*23 points total, allocation noted below*)

This question will use the functions **palindrome** and **reverse** as described in pages 388-396 of your text.

HOWEVER, the code in the text is not written using current ANSI standards. Therefore, more up to date versions of the source code (that will compile better on the ***acadunix*** system) are linked in WBW section 6.4.2. Please download and use the more current versions of the code.

First write a new **main** function and place it in a file called **prompt.c**.

The new **main** function will prompt the user for a string and then use the **palindrome** function to determine if the string is a palindrome.

**NOTE:** Function **main** in file **prompt.c** will call function **palindrome** in file **palindrome.c**. Function **palindrome** calls function **reverse** in file **reverse.c**.

**prompt.c** should output:

string IS a palindrome

if the string is a palindrome

string is not a palindrome

if it is not a palindrome

Example program run:

**Enter a string:** dad

**dad IS a palindrome**

* 1. Turn in a listing of your **prompt.c** source code. (*7 points*)

listing of prompt.c:

#include <stdio.h>

#include "palindrome.h"

int main(void);

/\*\*\*\*\*\*\*\*\*\* main function \*\*\*\*\*\*\*\*\*\*/

int main() {

char userString [100];

int isPalindrome = 0;

printf("Enter a string: ");

scanf("%s", userString);

isPalindrome = palindrome(userString);

if (isPalindrome == 1) {

printf("%s IS a palindrome\n", userString);

}

else {

printf("%s is not a palindrome\n", userString);

}

}

Separately compile the **reverse.c**, **palindrome.c**, and **prompt.c** using the either the **cc** or **gcc** compiler (whichever is available on your system).

* 1. What commands did you use to separately compile? (*3 points*)

cc -c reverse.c

cc -c palindrome.c

cc -c prompt.c

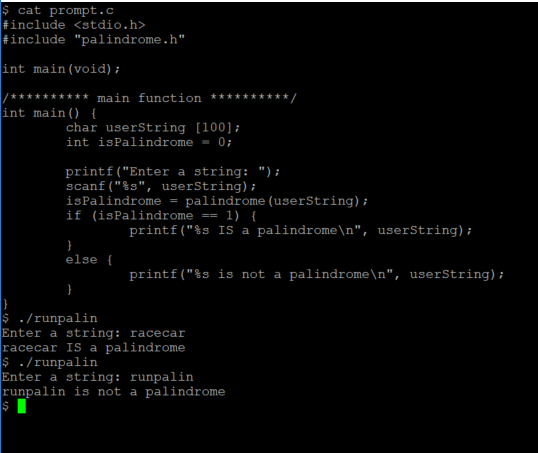
Then link **prompt.o** with **palindrome.o** and **reverse.o** into an executable called **runpalin**.

* 1. What command did you use to link and create the executable? (2 *points*)

cc reverse.o palindrome.o prompt.o -o runpalin

* 1. Finally, execute the program. What command did you use? (*1 point*)

./runpalin



Create a makefile that will separately compile all three files (prompt, palindrome, and reverse) and then link them to create the **runpalin** executable. Include all dependencies.

* 1. Turn in a listing of your make file. (*6 points*)

listing of make file named makepalindrome:

runpalin: prompt.o reverse.o palindrome.o

cc reverse.o palindrome.o prompt.o -o runpalin

prompt.o: prompt.c palindrome.h

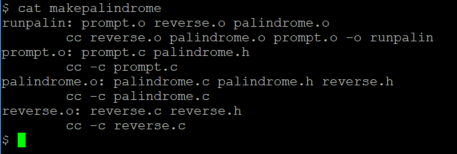
cc -c prompt.c

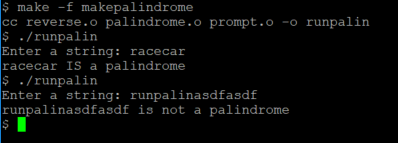
palindrome.o: palindrome.c palindrome.h reverse.h

cc -c palindrome.c

reverse.o: reverse.c reverse.h

cc -c reverse.c





* 1. Run the **touch** utility on all **.c** source code files.

What command did you use? (*2 points*)

touch -m \*.c

* 1. Run the makefile to re-create the **runpalin** executable file.

What command did you use? (*2 points*)

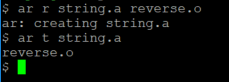
make -f makepalindrome

6. The **ar** archive utility (*12 points total, allocation noted below*)

Use **ar** archive files (NOT **tar** archive files) to do the following:

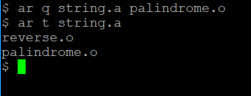
a) Create a new archive file called **string.a** and insert the file **reverse.o** into it.

What command(s) did you use? (*2 points*)  
  
ar r string.a reverse.o



b) Perform a quick append to place **palindrome.o** into the **string.a** archive file.

What command did you use? (*2 points*)  
  
ar q string.a palindrome.o



c) Display the archive file's table of contents. (*2 points*)

What command did you use?

ar t string.a

What was output? (supply a listing of the output)

reverse.o

palindrome.o

d) Create a makefile for **runpalin** that will compile and link **prompt.c** with the **reverse.o** and **palindrome.o** functions that are in the **string.a** archive.

Include all dependencies. (*6 points*)

To insure your makefile is working correctly:

- Run the **touch** utility on all **.c** source code files again.

- Run the new makefile to re-create the **runpalin** executable file.

- Execute **runpalin** and verify that it runs correctly.

Turn in a listing of your makefile.

listing of make file named makepalinar:

runpalin: prompt.o string.a

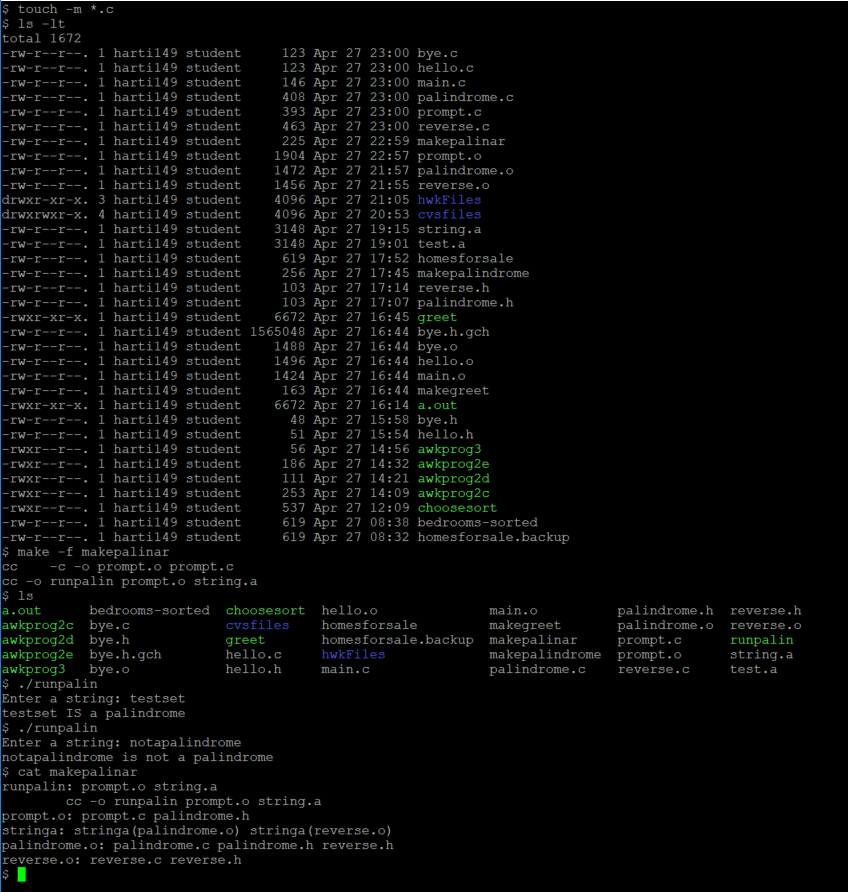
cc -o runpalin prompt.o string.a

prompt.o: prompt.c palindrome.h

stringa: stringa(palindrome.o) stringa(reverse.o)

palindrome.o: palindrome.c palindrome.h reverse.h

reverse.o: reverse.c reverse.h



7. The CVS utility (*10 points total – points as marked*)

Use CVS to complete the following tasks. (Use the **man** pages as necessary. )

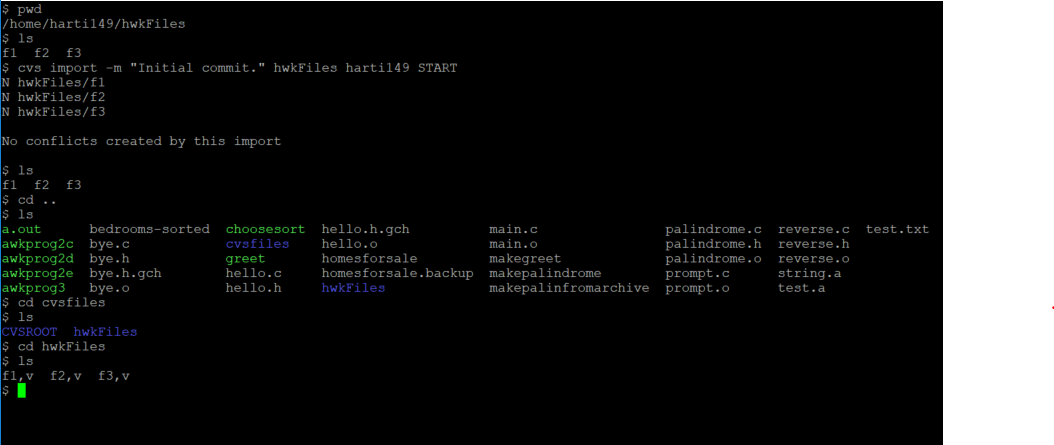
Before beginning this exercise, create a subdirectory in your home directory called **hwkFiles**, and place three files into it: **f1**, **f2** and **f3** (the contents of the files do not matter for this exercise).

a) Create a CVS repository named **cvsfiles** under your home directory. What command did you use? (*1 point*)  
  
cvs -d ~/cvsfiles init

b) Add the necessary line(s) to your **.profile** file, in order for all the CVS commands to work. What line(s) did you add? (*1 point*)  
  
export CVSROOT=$HOME/cvsfiles

c) Put all files in the **hwkFiles** directory into a CVS module, also called **hwkFiles**. What directory do you need to be in to do this? What command did you use? (*1 point*)  
  
You need to be in the same directory that has the files that you want to add to the CVX module.

cvs import -m “Initial commit.” hwkFiles harti149 START



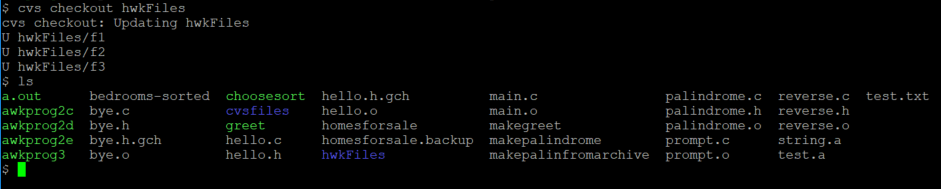
d) Delete your original **hwkFiles** directory, and all files in it. Why did you need to do this? (*1 point*)

rm -r hwkFiles  
  
We need to delete the original hwkFiles directory and contents so that it can be replaces with the version controlled directory / files of the same name(s) without conflict.

e) Create a working copy of the CVS module **hwkFiles** in your home directory. What directory do you need to be in to do this? What command did you use? (*1 point*)

cvs checkout hwkFiles

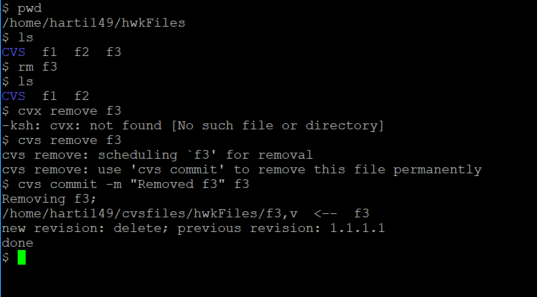
You need to be in your home directory.



1. Remove the **f3** file from your working directory and the CVS repository. List all commands you used. (*2 points*)  
     
   rm f3

cvs remove f3

cvs commit -m “Removed f3” f3

1. 

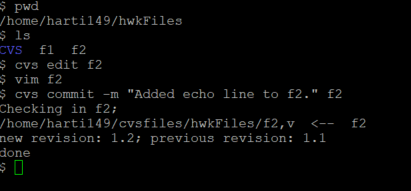
g) Let CVS know that you want to edit the **f2** file. Then edit the file. Afterwards, make sure the repository copy contains all changes. What directory do you need to be in to do this? List all commands you used. (*2 points*)

You need to be in your working directory again, where the checked out copie(s) of the repository files are.

cvs edit f2

vim f2

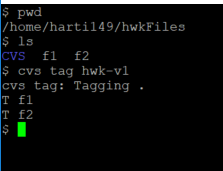
cvs commit -m “Added echo line to f2.” f2



h) Save the current version of the **hwkFiles** project under the tag **hwk-v1**. What directory do you need to be in to do this? What command did you use? (*1 point*)

You need to be in your working directory again, where the checked out copie(s) of the repository files are.

cvs tag hwk-v1



# Submission

This homework assignment is due by by **midnight Sunday of week 7**.

Submit a filled in copy of this Word document to the **Homework Assn 6** drop box (located under the Dropbox tab in the online course).

Before submitting the Word file with your answers, you MUST rename it as follows:

### Lastname-hwk6.docx

For example:

### Smith-hwk6.docx