All About Homework

The homework will consist of both programming exercises and written questions. This handout describes some general thoughts and techniques for doing homework, as well as what is required, how to submit it, how late homework is handled, and other administrative matters.

Structure of the Homework

Each homework assignment will have four parts: questions about Linux, questions about the C Programming Language, Programming, and Debugging.

Linux

The CSIF uses the operating system Linux. This system is widely used in computer science. Many large corporations (and small organizations) use it, or a very similar system called the UNIX system, for their servers and other systems. In this class, we will use this system and require that all programs must run on it.

These questions ask you about the Linux operating system. They will typically be short-answer questions. They may require you to include output from running a command. If so, please include a screen shot showing the result of the command's execution.

You are to put all the answers into a single file called "Linux.txt" or "Linux.pdf" (depending on whether you use text or PDF; see below). If you have screenshots and do not embed them in the submitted file, put a reference to them in that file and submit them as separate files named "Linux*n*.*ext*", where *n* is an integer and *ext* is an extension indicating the type of image file (like "jpg", "tif", or "gif").

C Programming Language

This will typically have 2 questions. You may do *either*; you are not required to do both.

One (or more) of the exercises will be problems in MyProgramming Lab. Please do these on MyProgrammingLab. When you do them, you will be told if your answer is correct. If not, you should try again; we will grade *only* the last one (which, we expect, will be correct!) Please turn in a file with base name "MyProgLab" (it does not matter what it contains or what the file name extension is) so we know to grade those exercises.

The other exercises will be review questions from the textbook. These will be either written answer or require a snippet of code. Unless the exercise specifically asks you to write or alter a program, please do *not* include a full program! Just include the snippet of code.

You are to put all the answers into a single file called "CPL.txt" or "CPL.pdf" (depending on whether you use text or PDF; see below).

Programming

This section will require you to write or modify one or more programs. These will be graded both for correctness and for style.

For correctness, please submit the program to the Gradebot (see below). The Gradebot will compile and run your program, and present you with the results of the test input. The results will simply say whether the program produced the correct output (or error messages) for the test. If you did not get all the tests right, then you can fix your program and resubmit it. The results shown on the Gradebot will form the correctness portion of your score.

For style, we look for well-commented, easy to read code. The document **Program Style** gives guidelines that you should follow. In particular, the guidelines about comments are *critical* because comments enable others to read and understand your code. And the "other" could be you, several weeks from now!

You are to submit the *final* version of the source code of your program to SmartSite. That file, or those files, are to be named as stated in the homework assignment.

Debugging

The debugging portion of the homework assignment requires you to fix a buggy program. The problems may be compiling problems (there will be errors, not warnings), execution problems (where the program runs but gives the wrong output), or both. As you debug the program, put comments into the source describing what changes you make.

Turn in the modified source file(s). If you are asked to use *gdb*(1), include a file showing the *gdb* commands you typed, the results, and what those results tell you. This file should be called "gdb.txt" or "gdb.pdf" (depending on whether you use text or PDF; see below). If you have screenshots and do not embed them in the submitted file, put a reference to them in that file and submit them as separate files named "gdb*n*.*ext*", where *n* is an integer and *ext* is an extension indicating the type of image file (like "jpg", "tif", or "gif").

Turning In Homework

All homework is due at 11:55PM on the due date, unless noted otherwise on the assignment. These will be graded and comments returned to you as quickly as possible; we'll try for three class periods, but can't guarantee it.

As stated above, all written homework must be submitted as text (ASCII) or as PDF (you can use any text processor you like to generate these). Please do not submit Microsoft Word (or similar) files; because the graders may grade these on UNIX-based and Linux systems, they will not be able to read those files. If your file is an ASCII file, please choose a name that ends in ".txt". If your file is a PDF file, please choose a name that ends in ".pdf". If you have screenshots or other images and do not embed them in the submitted file, put a reference to them in that file and submit them as separate files named "*basen*.*ext*", where *base* is the base name of the file, *n* is an integer and *ext* is an extension indicating the type of image file (like "jpg", "tif", or "gif").

For programs, turn in the source code file(s). As we will grade them using systems in the CSIF, please check that your programs run on one of the Linux systems there.

Please turn in your written exercises and programs electronically through SmartSite. If you need to turn in something on paper (for example, a diagram that you can't draw using your text processing program), please hand it to the professor or TA before the assignment is due, and put a note in what you submit electronically that you have done this. (That way, the reader will know to look for something written, rather than mark you off for that problem.)

Doing Written Exercises

When you are asked to analyze something, or explain something, write clearly and cogently and use proper English — if we can't understand it, it's wrong. If the question asks for an opinion, state your opinion clearly, justify it, and don't ramble. Answers that start, "My opinion is yes ..." and conclude with "...on the other hand it could equally well be no" won't get much credit.

Doing Programming Assignments

Please do not leave assignments for the last minute. The assignments are non-trivial and will require significant design time before you start programming and debugging. When we decide on the due dates, we assume you will spend significant amounts of time on design as well as coding and debugging. If you choose not to do this, you will have difficulty finishing the assignments on time.

Please take the time to design your program carefully. More programming problems arise from improper design than anything else, and the few hours you spend on design will be amply repaid by shorter coding and debugging phases. So please think the design and interfaces through, and — as always — try to find the simplest way to do the assignment (within the limits given in the assignment, of course)!

We do not mind being asked for help; indeed, we welcome it because it helps us know what students are finding difficult or confusing, and sometimes a few words about the problem in class will clarify the assignment immensely. We do mind being asked for help before you have tried to think the problem through. The classic objectionable question (this really happened) occurred on a homework assignment in which the class was given a buggy program. The assignment said the program did not work, and the homework was to debug it and make it work. That particular class period discussed how to deal with bugs, and gave tips and techniques on how to debug programs. Within 10 minutes of the end of the class during which the assignment was given out, the instructor got this request for help: "The program doesn't run. What do I do now?"

So, before asking for help, please be sure that you have:

1. spent a significant amount of time on the design of your solution;
2. used a debugger if the problem is a programming bug;
3. read all relevant handouts, and news articles (because your question may be answered there); and
4. tried everything you could think of to solve the problem.

When you come to us, or send us a note, asking for help, please show us whatever you have done to solve the problem, because the first question we will ask you is "What have you tried?" This isn't because we think you're wasting our time. It's because understanding how you have tried to solve the problem will help us figure out exactly what your difficulty is and what we can do to help you. Remember, we will do everything we can to avoid solving the problem for you. When we give you help, our goal is to help you solve the problem yourself.

Late Homework and Grade Reviews

In general, late homework will not be accepted. Requests for exceptions will be handled on a case-by-case basis; for example, medical reasons tend to get extensions. So please do feel free to ask!

If you feel that there is an error in grading, please come see a TA and we'll look over it (and possibly talk with you about it). However, don't dally; any such request must be made within one week of when the grades were made available. After that, we won't change your grade.

The Gradebot

The Gradebot is a system developed by Prof. Hao Chen to grade your programs automatically. It allows repeated resubmissions, so you can submit your program for grading multiple times. We will use the *last* score you get on that system. Here, we show you how to submit your program and look at the results. But first, you have to register on the Gradebot.

Step 1. Registering on the Gradebot

Every student on the roster in this class has been added as a student. Go to `https://gradebot.org/gradebot` and use your UC Davis email address (you*must* use this!). Then follow the instructions. Once you have registered, log in and complete your profile. The critical part here is to add your student ID, because we will use that to track scores. In other words, if it isn't there or is wrong, there is a very good chance you won't get credit for your scores. Also, be aware there is a feature that allows everyone in the class to see your screen name, avatar, and score, but *not* your student ID or user name. So do not choose your *real name* or *UCD email* as your screen name, or your photograph as your avatar, unless you want people to know how you are doing on the scoreboard!

Once you register, you will receive email from the Gradebot with a "private key". You will use this to submit homework. This private key will be in a file attachment called "id_dsa". When you get this key, do the following:

1. If you do not have a directory called ".ssh" in your home directory, you must create it. Go to your home directory and type `ls -a`. If you see ".ssh" in the output, you have one. If not, create it by typing:

   ```
   % mkdir .ssh
   ```

   The "." in front of the "ssh" is critical — leave that out, and nothing will work.
2. Put your private key into this directory. From the directory where you downloaded the key, type

   ```
   % cp id_dsa ~/.ssh
   ```

3. Finally, set the access controls for this directory, and all the files in it, so that only you, the owner, can read or write the file. Type

   ```
   % chmod -R go= ~/.ssh
   ```

   The option "-R" means to apply the command recursively, that is to the directory you name, all the files and subdirectories in it, and all their files and subdirectories, and so forth. The "go=" means set the "g" (group) and "o" (other) permissions to nothing (the "=" with nothing after it).

Now everything is set up and you're ready to get to work on the programs.

Step 2. Setting Up Your Local Workspace

Using the Gradebot involves a program called *git*(1). You don't have to know how it works or what it does as long as you follow these steps.[1]

1. Create the source code directory by this command:

   ```
   % git clone metastasis@gradebot.org:user/your_username/1/hw_num
   ```

   where *your_username* is your UC Davis email id (without the "@ucdavis.edu") and *hw_num* is the homework assignment number. For example, the first homework program has a *hw_num* of 1. So, when I want to create the directory, I would type:

[1]Oh, all right. It is a source code control system that keeps track of the versions of your program. Basically, you create and edit your program on your computer. You then transfer your program to a remote *git* repository. As you update your program and transfer the newer versions to the repository, *git* keeps track of the changes. That way, you can always "go back" to an earlier version.

```
% git clone metastasis@gradebot.org:user/mabishop/1/1
Cloning into '1'...
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.ISO8859-1)
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
LANGUAGE = (unset),
LC_ALL = "en_US.ISO8859-1",
LANG = "en_US.UTF-8"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 20 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (20/20), done.
Resolving deltas: 100% (5/5), done.
Checking connectivity... done.
```

As you can see, I will get several warnings (which I, and you, should ignore). When the command finishes, I will have a directory called "1"
If you are asked for a password, you didn't put the file "id_dsa" in the right place. Go back to that step, and be sure the file goes into the ".ssh" subdirectory of your home directory.

Step 3. Submitting a Program for Grading

Now the hard part. I write a version of the program and submit it to the Gradebot for testing. Here, I will assume you have written the program and it is in a file called "prog.c" and it is in the directory "1" that you created in the previous step. You will then submit it to the Gradebot for testing, as follows:

1. First, tell *git* about it:

   ```
   % git add prog.c
   ```

   Unless something goes wrong (like you mistyped the file name), this command should give no output.
2. Next, check the program in:

   ```
   % git commit -a
   [master 2f1eef7] minor comment change for demo purposes
    Committer: Matt Bishop <bishop@pc24.cs.ucdavis.edu>
   Your name and email address were configured automatically based
   on your username and hostname. Please check that they are accurate.
   You can suppress this message by setting them explicitly. Run the
   following command and follow the instructions in your editor to edit
   your configuration file:

       git config --global --edit

   After doing this, you may fix the identity used for this commit with:

       git commit --amend --reset-author

    1 file changed, 1 insertion(+), 1 deletion(-)
   ```

   This will start a text editor, and the window will have several lines of text beginning with "#". Add a brief comment *after* the last line saying what you have changed. Then save the file.[2] If you see output similar to that above you can safely ignore it (it's a configuration suggestion, not an error).

[2]The default editor is *vim*(1). To add a comment, when the editor starts, type `L` to move the cursor to the last line, `a` to add your comment after the last line; then type your comment. When you are done, type the `ESC` (escape) key to end the comment, and then `ZZ` to save the file. This takes you back to the shell.

3. Now send your changes to the remote repository:

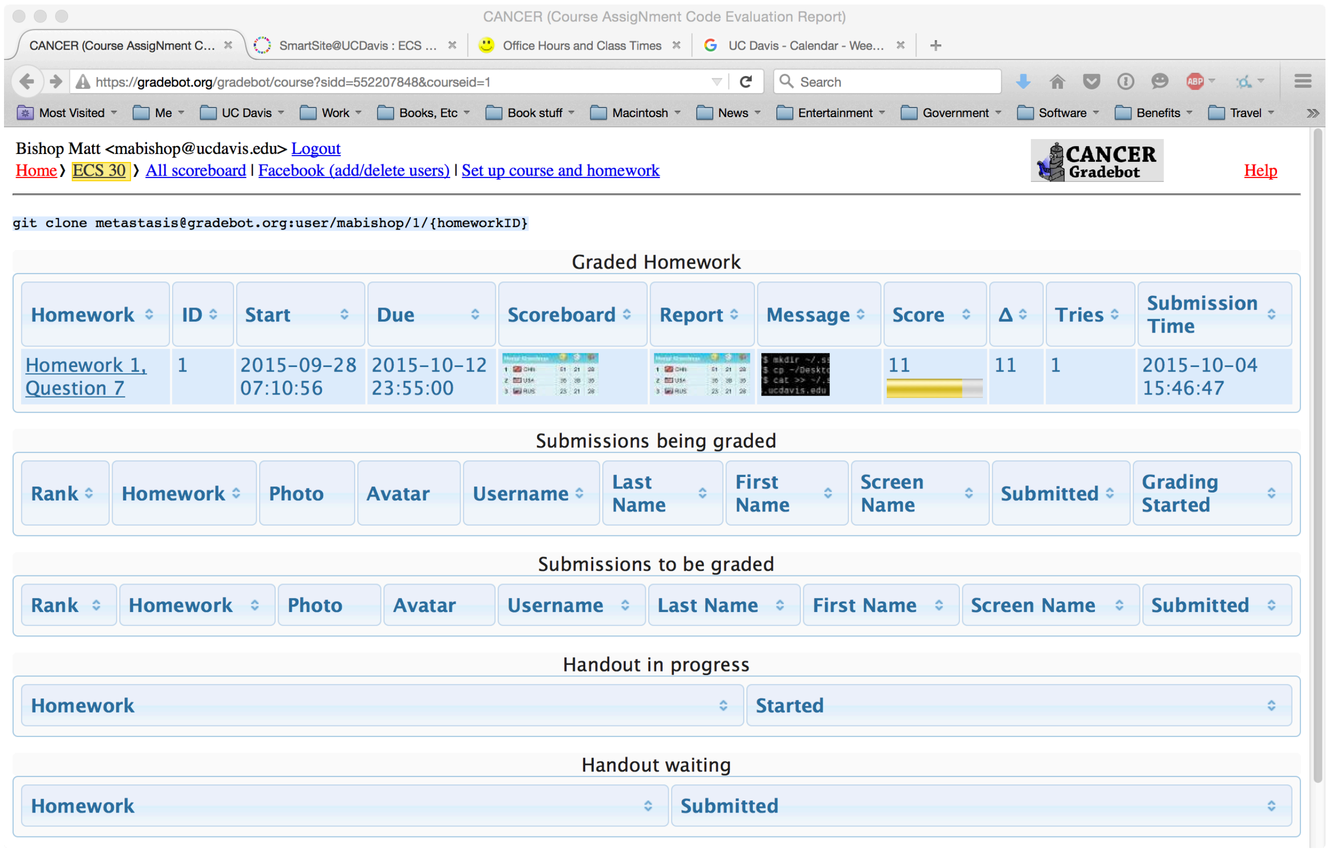
```
%git push origin master
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.ISO8859-1)
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
LANGUAGE = (unset),
LC_ALL = "en_US.ISO8859-1",
LANG = "en_US.UTF-8"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: perl: warning: Setting locale failed.
remote: perl: warning: Please check that your locale settings:
remote:  LANGUAGE = (unset),
remote:  LC_ALL = "en_US.ISO8859-1",
remote:  LANG = "en_US.UTF-8"
remote:     are supported and installed on your system.
remote: perl: warning: Falling back to the standard locale ("C").
remote: Gradebot accepted your submission. Good luck.
To metastasis@gradebot.org:user/mabishop/1/1
   f9cc30f..2f1eef7  master -> master
```

The Gradebot will automatically run your program.

Step 4. See How You Did

Go to `https://gradebot.org/gradebot`, click on "ECS 30", and look at the "Graded Homework" section. Your assignment should be there. Look over at the "Score" column. The horizontal bar in that entry should be green and extend all the way across the entry. That means you got all the test cases right (good job!). If it doesn't, the bar will be yellow or red, indicating you missed some tests.

Here is the result of my first try at writing the program for the first homework.



Bishop Matt <mabishop@ucdavis.edu> Logout

Home ⟩ ECS 30 ⟩ All scoreboard | Facebook (add/delete users) | Set up course and homework

Help

```
git clone metastasis@gradebot.org:user/mabishop/1/{homeworkID}
```

Graded Homework

Homework	ID	Start	Due	Scoreboard	Report	Message	Score	Δ	Tries	Submission Time
Homework 1, Question 7	1	2015-09-28 07:10:56	2015-10-12 23:55:00				11	11	1	2015-10-04 15:46:47

Submissions being graded

Rank	Homework	Photo	Avatar	Username	Last Name	First Name	Screen Name	Submitted	Grading Started

Submissions to be graded

Rank	Homework	Photo	Avatar	Username	Last Name	First Name	Screen Name	Submitted

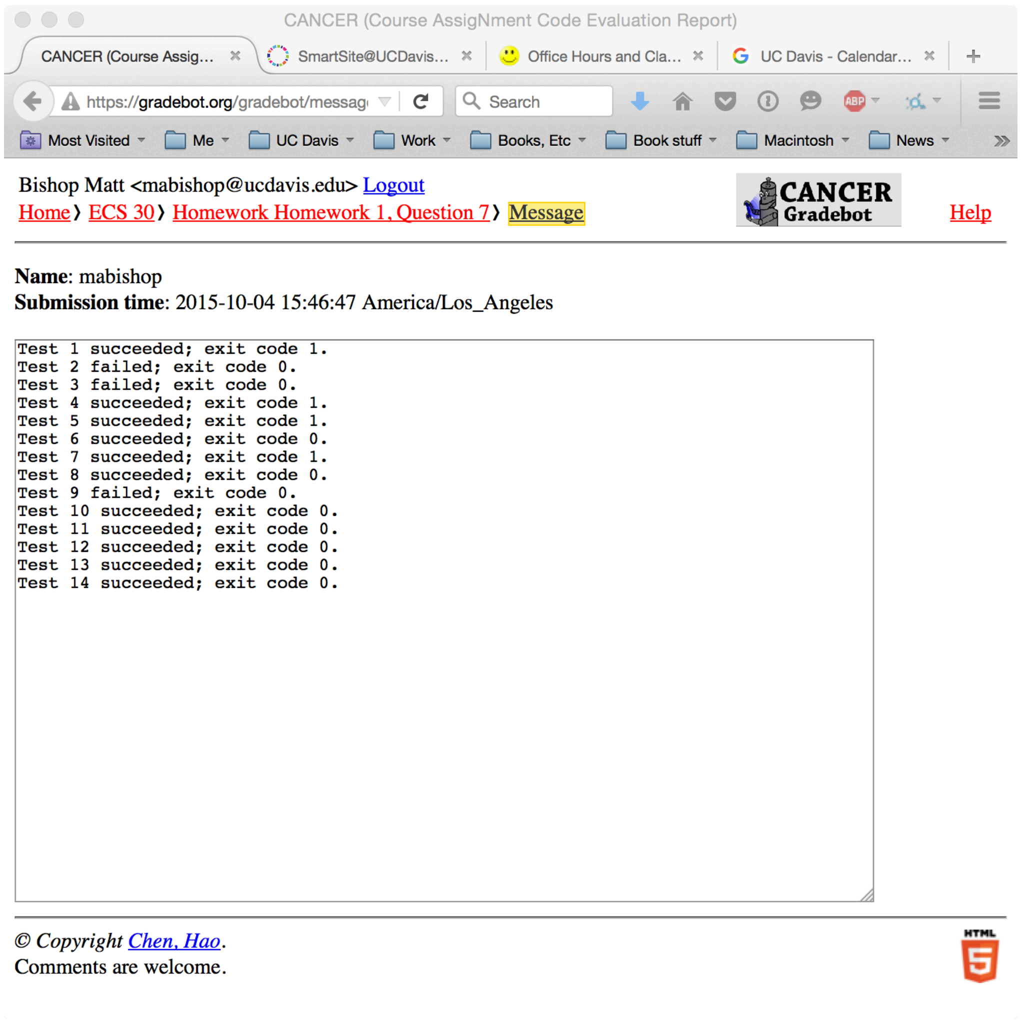
Handout in progress

Homework	Started

Handout waiting

Homework	Submitted

Note I got 11 test cases correct on this, my first try, but the bar is yellow and does not go completely across the entry. So I missed some. Now I want to see which test cases I missed. I click on the Messages entry (which is a dark image) and the following comes up:



Bishop Matt <mabishop@ucdavis.edu> Logout
Home ❯ ECS 30 ❯ Homework Homework 1, Question 7 ❯ Message

Name: mabishop
Submission time: 2015-10-04 15:46:47 America/Los_Angeles

```
Test 1 succeeded; exit code 1.
Test 2 failed; exit code 0.
Test 3 failed; exit code 0.
Test 4 succeeded; exit code 1.
Test 5 succeeded; exit code 1.
Test 6 succeeded; exit code 0.
Test 7 succeeded; exit code 1.
Test 8 succeeded; exit code 0.
Test 9 failed; exit code 0.
Test 10 succeeded; exit code 0.
Test 11 succeeded; exit code 0.
Test 12 succeeded; exit code 0.
Test 13 succeeded; exit code 0.
Test 14 succeeded; exit code 0.
```

© Copyright Chen, Hao.
Comments are welcome.

This says I got all but tests 2, 3, and 9 right.

As programs are run, the Gradebot makes available a scoreboard that contains your rank, avatar, screen name, messages, score, change, number of tries, and the time of last run. All registered students can see this.

This is why you must choose your screen name and avatar carefully. In particular, if you want privacy, make them something not connected with you.

Step 5. Revise and Resubmit

If you got all the cases correct, you're done! Otherwise, figure out what is wrong with your program, change it, and repeat steps 2–5. You can repeat these steps as many times as you like; we'll only take the last score.

Important Reminder

Remember to submit the final version of your program through SmartSite!