**Operating Systems CSY2002**

**Assignment 2 Resit**

**Linux Programming**

Date: 14/6/2019

**Question**

Write a shell script (using the *tar* command) which compresses, zips and archives all the files in a chosen directory which have been modified since they were last archived. You will need to use crontab. You will need a menu screen which allows the System Manager to select the day, time and frequency when he/she wishes to archive the files and the source and destination locations. The script should send a message to the screen to say which files have been updated successfully or an error message if it didn’t. A record of these updated files should be stored in a file which can be overwritten at each update. You will need to show the edited crontab file and the program snippet you created to test it.

Name the program archive.sh.

Refer to the section named “*How to Setup a Crontab File*” on pages 4 and 5 of this document for information about setting the date and time for archiving.

***Remember to use regular expressions for error checking. i.e when you prompt the user to enter a value or name you should test to see if it is in the correct format.***

Deliverables - you will need to include the following:

* Program code using functions which is correctly commented and indented with error checking of all inputs.
* Flowchart.
* Implementation – this is a description of how the code works.
* Test plan and evidence of testing (screenshots– this should show your personalised prompt).

The program should be clearly commented as outlined in the handout marked “*How to document shell scripts* “on pages 6 and 7 of this document. The program should contain comprehensive error checking where appropriate and screenshots which show the path, including your student ID or name in the address bar. ***If I can’t identify that the screenshots come from you, I will assume they are not your own work and you will therefore not be given credit for them***. These screenshots will include tests showing the outputs of all scenarios. e.g outputs when incorrect data is entered by the user. There should be an accompanying report containing a description of the programs including flow charts. Flowchart symbols may be obtained from Google Docs.

The highest grades will require the programs to be well written. This means the inclusion of functions, meaningful variable names, indentation etc.

|  |  |  |
| --- | --- | --- |
| **ASSESSMENT ITEMS** | **WEIGHTING** | **LEARNING OUTCOMES** |
| Code as per guidelines | 55% | c ,f, g, i, j |

Fill in the Assignment Cover Sheet (attached) with your name and student number.

Brian Thompson

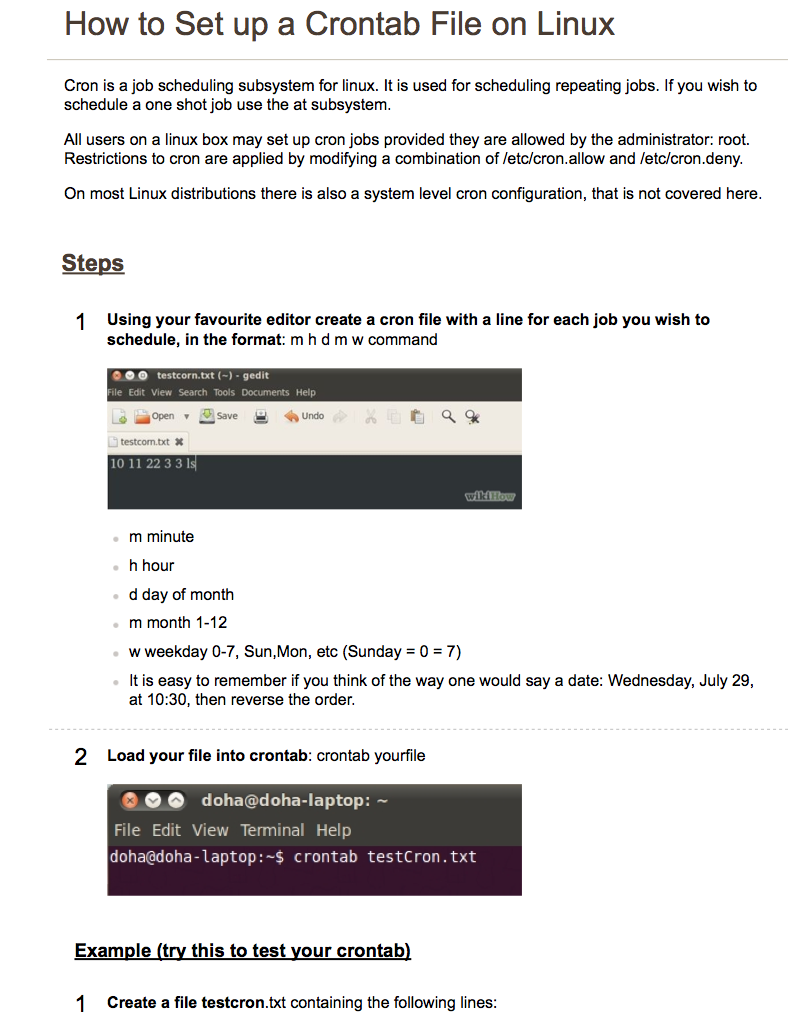
**Hand in Date: Friday 12/07/19**

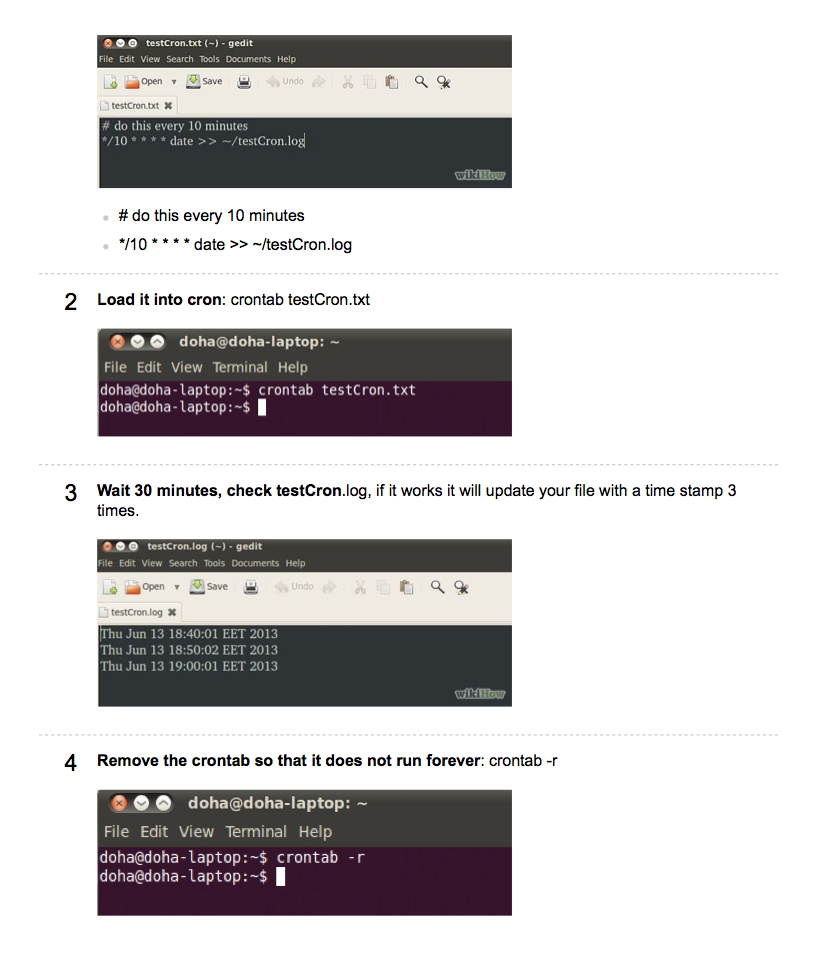
|  |  |  |  |
| --- | --- | --- | --- |
| BSc Computing  **CSY2002 Operating Systems** | | | |
| Date of Issue: | 14th June 2019 | Date for  Submission: | Friday 12th July 2019 |
| Agreed Date for late submission: |  | Module Tutor:  Signed: | Brian Thompson |
|  |
| Student Name:  Student ID: |  | | |
|  | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student’s**  Signature: |  | | | | | |
| This assignment is weighted as 50% of the Module's assessment | | | | | | |
| **Assessment Feedback** | | | | | | |
| Aspect | | Excellent | Good | Satisfactory | Needs some more work | Needs much more work |
| Code correctness and functionality (40%) | |  |  |  |  |  |
| Code layout – indenting, comments, use of functions etc (10%) | |  |  |  |  |  |
| Flowchart (10%) | |  |  |  |  |  |
| Implementation (20%) | |  |  |  |  |  |
| Testing including evidence(20%) | |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Specific aspects of the assignment that the marker likes: | | Specific aspects of the assignment that need more work: | | | |
| **Tutor’s** Signature: |  | Date: |  | Grade: |  |

The University of Northampton’s Policy on Plagiarism & Mitigating Circumstances will be strictly implemented. By submitting this **signed** assignment you are asserting that this submission is entirely your own individual wor





**How to Document Shell Scripts**

The two golden rules when writing shell scripts are:

1. Whenever possible, test the commands you want to use outside of your shell script. Test and perfect your commands at the command line.
2. **Do some design!** Think the program through before you begin writing your code.

There are two general types of people who will read your shell scripts, namely:

* *The user of the shell script* and *someone who may want to modify it*.

The user of a shell script needs to know **how to run** the script, what it needs and **what it does**. Thus the beginning of your shell script should have the following information:

* A general statement about what the shell script does. Notice this is **what** it does, not **how** it does it.
* How to run the shell script, including any information on options and arguments.
* Any active shell script needs. This includes whether the shell script functions in command or interactive mode, and any input the user must provide.
* What output the shell script generates and where it goes.
* Any assumptions the shell script makes about its environment. For example, if your shell script expects to be able to write to the current directory, you should say so. If it needs to use standard data from a particular place this must be documented, including where it must be, and its format.

The goal of documentation in the program header is to get the user to read it! This means that too much documentation is as bad as insufficient documentation.. Avoid documenting things that are obvious. Keep documentation concise and organise it so that it is easy to read, with the most important points at the beginning or otherwise highlighted.

The second audience is a person who may **wish the modify it**. This person will be a Unix programmer with expertise as extensive as yours. This person needs to know how your program does what it does. This type of documentation needs to be in the body of the code, with the pieces of code it documents. Remember, This person has expertise, so the only things necessary to document are things that are not completely obvious. If you are using a filter such as cut, don’t explain how it works. A comment reminding the reader the contents of the fields you are cutting would be more helpful.

Documentation in the body of the code (called *inline documentation*) should not be added to the right of the code. Instead add entire lines of the documentation above the section of the code being documented.

Although it should be obvious, ensure that your documentation is correct.

Documentation is so important it will be worth 15% of the grade of each program.

**Style**

Your shell script should be easy to read. This means that the flow of control should be obvious, the names of the variables descriptive and the design as simple as possible.

Proper style requires the use of consistent indentation and white space.

You should also ask yourself the following questions:

* Is your solution well designed?
* Did you complete one part of the task before starting another?
* Are your error messages and other output done consistently and are they understandable?
* Did you give the user sufficient information to readily correct the problem?