# CST8177 – Assignment 3
## Lock/Unlock Accounts

# Objectives

1. To understand how to  plan and design a system administration script;
2. To design a Test Plan for testing the script to ensure that it operates correctly, and doesn't allow erroneous actions;
3. To write a brief User's Guide for the completed script;
4. To understand how to  write and debug a script;

# Submission

This project requires only one submission, a paper-based submission in your lab teacher's physical dropbox. The submission has to be stapled and need not be submitted in an envelope.

The submission contains:

- Cover page listing course information, lab information and student information (see the submission standard on the course web site).
- Table of contents.
- The following analysis and design files:
    - Your restatement of the problem: the Requirement.
    - Your Analysis of the problem for your proposed solution, consisting of at least:
        - Lists of all your **`stderr`** messages and **`logger`** output and the conditions where they can occur.
        - Samples of your normal **`stdout`** messages, including prompts for input.
        - Data Dictionary for each script or function's key variables;
        - PDL for each script or function.
    - A brief description for a system administrator to use your script (How-To or User's Guide – only a page or so);
- A complete, commented, and properly indented printed copy of your main script, and all functions or source files necessary to execute it without error.
- Your Test Plan for confirming correct operation, and demonstrating the detection of errors. Each test will consist of a test id, the exact input to be used, the (brief) purpose of this test, and the expected result
- Your Test Plan results (a typescript from **`script`**) for confirming correct operation, and demonstrating detection of errors, plus:
    - Test input files and the portions of system files for added test users.
    - If you build testing scripts, provide them and the design and data files.

# Plagiarism

Plagiarism will not be tolerated. College plagiarism policy will be strictly enforced, although you may submit your solution as a team of two. You can even work in a larger group, but each pair or individual must prepare their own solution and submit only their own work. Luckily (for me), copied code is not difficult to identify. It's almost as though you leave fingerprints or DNA fragments on your work.

# Background

## Problem description

For this assignment, you will be the system administrator of a fictional company called AlgoTech, a medium-sized software development company. As such, you are in charge of the user accounts on the company-wide computer complex, which is implemented as a Fedora Linux server cluster.

Your supervisor has realized that for some business reasons, a large amount of sysadmin time is taken up with locking and later unlocking user accounts. She asks you to develop a simple script to improve the automation of lists of accounts, making use of the **`passwd -l`** and **`passwd -u`** commands.

## Proposed solution

There will be a single script used as the CLI tool (note: you may choose to build the solution as several scripts or functions to minimize duplicate code, but the user will only operate with one apparent script) named **`lock`** (to lock accounts) with a hard link with the name **`unlock`** (to unlock accounts). Both entries will reside in and run from the directory **`/usr/local/sbin`**.

The command line for this script will support all of these three forms:

1. Prompt for and read in any number of account ids from **`stdin`** (one per line) to be locked or unlocked; detect end-of-file (Control-D) in order to terminate the script. This is the no arguments case.

```
xxx# lock
Enter Account: user123
Enter Account: ^D
Accounts user123 locked
xxx#
```

2. The **`filename`** (and optional path, absolute or relative) given is read and each user account id (one per line) is locked or unlocked. This is the exactly 2 arguments case.

```
xxx# lock -f accounts.text
Accounts user567 user678 locked
xxx#
```

   where **`accounts.text`** is:

```
user567
user678
```

3. Use any number of arguments supplied as account ids to be locked or unlocked. There is no practical limit to the number of account ids that can be supplied as long as the command line does not exceed 256 characters total. This case has at least 1 argument.

```
xxx# unlock user123 user234 user345 user456
Accounts user123 user234 user345 user456 unlocked
xxx#
```

**Note these restrictions:**

- The only user who can use this script is **root**. Reject any other user.
- An account must be valid. That is, there must be a valid entry in the password file for each account id to be locked or unlocked. Each account must have a valid password.
- The account must not already be locked (for **lock**) or must already be locked (for **unlock**) prior to the script's action.
- Ensure that each error is logged to **stderr** and also to the log file **/var/log/acct-lock**. Each successful **lock** or **unlock** must also be recorded in **/var/log/acct-lock**, each entry marked with a distinctive tag for ease of searching for entries (locked OK, unlocked OK, locked error, unlocked error, etc.) suitable for the production of a weekly report (no reporting script is required in this assignment).

## Sample Logic Summary

You may use this as an outline of the logical for solving this problem, or you may create your own (Steps 1 to 3 from Section A of the Method below).

- Reject user if not **root**
- Use command-line arguments to determine input method:
  - If no arguments, read list of accounts one at a time from **stdin**
  - If 2 arguments and the first is **-f**, read the list of accounts one at a time from the second argument as a file
  - Otherwise, use the arguments as the list of accounts
- In each case, process each item of the list of accounts as follows:
  - Make sure account exists
  - Examine password for valid-locked or valid-unlocked
  - If command is **lock** and password is valid-unlocked, lock it
  - If command is **unlock** and password is valid-locked, unlock it
  - Issue messages and log information as required.

# Method

## Section A – Analysis and Design

Use the method provided below to work through the steps for solving this problem:

1. Write down all the things your script has to do in a point-form list;
2. Re-arrange them as necessary to get a reasonable sequence;
3. Add in details about tests and loops;
4. Identify all the messages and prompts you need and where to read any answers, and determine the structure of any data files you need;
5. Walk through your refined list and check it against the problem as given - you may have to add further details or re-arrange items;
6. Now write the PDL in good pseudocode from the list in step 5;
7. Walk through the pseudocode and check it for accuracy and completeness, repeating steps 6 and 7 until you're satisfied;
8. Make a list of all the variables you will need and what data they will contain Data Dictionary), and create your data files from step 4;

## Section B – Building the script

Use the method provided to write and test the script:

1. Write your script from the PDL, a few lines at a time and not the whole script at once, and test it at every step. It is much simpler to test and correct a script a few lines at a time than it is to write a long script and debug it all at once.
    a. Use the final PDL and information from your Data Dictionary and input/output text to translate every few lines into script. You may wish to leave in your PDL statements as comments;
    b. Test your script so far (small chunks of code are easier to check than big blocks) - you may need to add script lines and **`echo`** statements to make your partial script work;
2. Repeat steps 1a and 1b until your script is complete;
3. Finally, test it thoroughly against the original problem statement (this is not the Test Plan).

Be sure to make frequent back-up copies while you're working.

## Section C – Testing and Using Test Results

Your Test Plan is not complete after you first create it during Analysis and Design. You will come across concerns and issues while you are writing your script and get an unexpected result. Go back immediately and update your Test Plan (and your PDL, if your logic changes).

Each test consists of a test id, the actual input, the purpose of this test, and the expected result. Start with test number 1, and continue test by test until the actual output does not match your expected output. Find and correct the defective part of the script (or correct this test if it's a Test Plan error). Resume at test 1, since you can't tell if new errors have been introduced.

Continue until you have completed a successful execution of every test. Use the **`script`** typescript command to make a file of this successful run of your Test Plan for inclusion in your submission.