Project 2: Version Control

Due Wednesday February 4 at 11:59pm

All projects in this course are submitted via CourSys(http://courses.cs.sfu.ca). Submit a ZIP file of all project deliverables via CourSys by the project deadline. For every 24 hours past the original deadline, 10% of the original score shall be taken away, up to a maximum of two days.

This assignment is to be done individually. Do not share your code or solution. Do not copy code found online. Do not post questions about the assignment online. Please direct all questions to the instructor or TA(cmpt-276-help@sfu.ca). You *may* make use of any code provide by the instructor, or in help guides/documentation provided by the instructor.

Git Basics

1) Create A Local Repository

You already have a working project from Project 1. Now you need to create a local repository for it. Android Studio helps to automate this process. Under the VCS menu on the menu bar, select Enable Version Control:

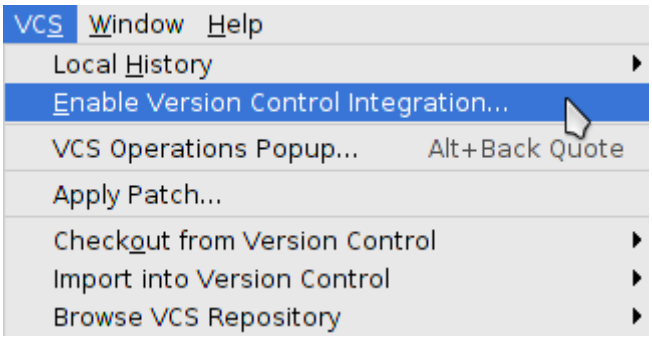

Figure 1: Enabling version control within Android Studio

A dialog box will pop up asking you which version control system to use. Select Git and continue.

2) Add your project files

Notice that most of the files in your projects are now colored red in the Project View (the left frame). This means that they are not yet tracked by Git. In order for Git to track their histories, you must identify exactly which files to track.

One easy way to do this is to use the menu item VCS→Show Changes View. The bottom frame of the IDE now shows the present status that Git reports for files. Clicking the triangle next to "Unversioned Files" shows all of the files that you want to add to the project. Select all of them at once, right click, and select "Add to VCS" fro the menu.

Select the files again, right click, and select "Commit Changes" to finish adding these files to the tracked history in Git. Notice the "Commit Message" portion of the dialog that pops up. This is where you should type a meaningful description of the changes that you are committing to the project. This time, just type "Initial commit of project files". Enter your name into the Author field as well and then continue.

3) Change and Commit

Add a new button labeled "Disclaimer of Liability" to the main screen of the application, just below the title. Stretch the button so that it fills the width of the screen. Notice that when you save the files, they turn blue within the Project View. This indicates that the files have changed, and those changes have not yet been committed to the local repository. Use VCS→Show Changes View and notice that it lists all of the files that have changed (in blue).

Now add a background image to the main activity of the project. Save an image in the `app/src/main/res/drawable-hdpi/` subdirectory of your project. You may use any image as the background, but your UI elements must still be readable. Notice that the image file name shows up as red in the project view because it is not tracked by Git. Right click it, find the Git menu, and choose Add.

Now commit your changes to Git using VCS→Commit Changes to commit all of the changes to the project as one event in the project history. Enter a meaningful commit message and continue.

4) Log Into Your GitLab Account and Create a Remote Repository

For this class, we'll be using a GitLab server to host remote repositories for our projects. GitLab provides a web interface to creating and managing projects using Git. Your account on the server will be created the first time that you log in. Go to the CSIL Git Server and log in using your CSIL username and password.

Now that you are logged in, you need to create a repository for your project. Click on the New Project button, which looks like a plus sign in the upper right corner of the page:

The following page lets you customize aspects of the project, such as its name, description, and who else can view or modify the project. Make sure that the

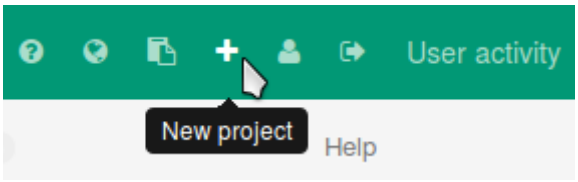

Figure 2: The New Project button in GitLab

Namespace for the project shows your name and that the *Visibility Level* for the project is Private. This will prevent other people from being able to access your code without your permission. When we have team projects, you can use different settings to allow only your team to access your code, as well. Click "Create Project" at the bottom to finish creating the remote repository.

This will take you to your project page in GitLab. Once you have pushed files to the repository, this screen will contain useful information. For now, notice that the upper right hand corner contains the address of the repository. Copy this address. You will use it shortly to enable using Git within Android Studio.

At this point, you may also need to configure GitLab to recognize your ID and computer by adding a security key. You can do this by clicking on the Profile Settings button (near the New Project button), clicking on the SSH Keys tab, and following the instructions for adding an SSH key.

5) Push to the Remote Repository

To push your project to the remote repository, you must first configure your local repository to know where the remote repository is. There are two ways to do this.

1) If we were starting a *new* project, we could have created our repository using GitLab first and then *clone*d a local copy of the repository using VCS→Checkout from Version Control→Git. This clone would automatically know about the remote repository.
2) Since we already had our project, we are instead manually telling the local repository where to find the remote one. This can be done on the terminal by changing into the project directory and using the command: `git remote add origin git@csil-git1.cs.surrey.sfu.ca:<username>/<repository>` Note that the address at the end of the command is the same one you copied at the end of creating the repository with GitLab.

Push your local changes to the remote repository from within Android Studio by using VCS→Git→Push. In the dialog box that pops up, click the check box at the bottom. This is only necessary the first time you push to a new repository.

If you now use VCS→Git→Pull to pull down any updates from the remote repository, you should see the the Pull Info frame in Android Studio report that no items were pulled because there haven't been any additional changes to the project.

You now know the basics of using Git with Android Studio. You can add and commit your changes to your project as you go. At the end, your Git log will need to show at least the changes made here.

Adding a Disclaimer

Backing an evil overlord could have negative consequences. Now would be a good time to get the disclaimer in your app working. That way people will know that there are risks in supporting the Daleks or the Cylons. Add a disclaimer screen (activity) to your project. Chapters 2 and 8 of the Android book have helpful information about adding and managing activities. You'll also find that Android Studio has many helpful right-click menus for performing tasks like this.

Disclaimer screen features:

1) Create a title that has text in the center and an icon on the left and right sides. You *must* use a *sublayout* to do this for the project. Either a horizontal linear layout or a relative layout is a good choice. Use the gravity property of the text or layout to create the appropriate look. If you want a line of text to break at a specific point, use a newline character (`'\n'`) in the string. The displayed text must come from `strings.xml`. You may choose the particular icon. You can find many free options on the Crystal Clear Wikimedia page.
2) Make the Disclaimer of Liability button on the main activity launch the disclaimer activity.
3) Below the title, add some descriptive text, as in Figure 3. Your text must be long enough to take up multiple lines on the screen and must contain at least your name. *Do not* put any sensitive personal identifiers such as your student number in the application. This text must be stored in `strings.xml`.
4) Add a TextView that shows how many times the application has been started. The text (string) that is displayed in this view must be in `strings.xml`. You will almost certainly want to use Java code to load this string from `strings.xml` and then concatenate the number of starts onto the string. Java can concatenate a string and a number such as:
   ```
   String message = "Hello" + 42;
   ```
   To load a string by resource number from `strings.xml`, use:
   ```
   String message = (String)getResources().getText(R.string.string_resource_name);
   ```
   When counting the number of times your application starts, you must use `SharedPreferences` to save the value across runs. Increment it each time

the application starts. Recall from Chapter 2 that `onCreate()` in the main activity executes once each time your app starts. You will again use `SharedPreferences` on the disclaimer screen to load this value. You can find additional information on `SharedPreferences` in Chapter 11 of the Android book. You may also find Dr. Fraser's video on SharedPreferences to be helpful.

5) When the user taps anywhere on the screen, have it close the disclaimer activity. Recall again from Chapter 2 that the `finish()` method can close an activity. Android calls the `onTouchEvent()` method on your activity when there is an unhandled touch event. Override this and add your own code to the method. In Android Studio, you can use Code→Override Methods... to find methods you may override and create a template implementation.
6) Commit your changes with an appropriate message. (VCS→Commit Changes)

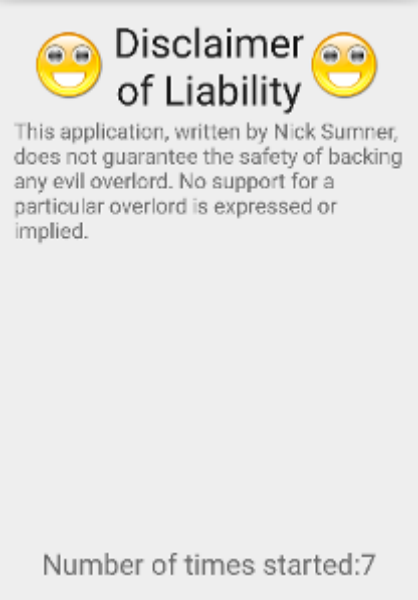

Figure 3: Example disclaimer screen. Your text may differ.

Vibrate

Have your phone rumble with terror when you show your support: use Android's vibrate feature whenever a Support button is pressed. When the user presses either of the Support buttons in the main activity, have the phone vibrate for 1 second (1000 ms). You will need to discover how to use the vibrate functionality

on your own. There are many excellent guides online. Usually, the first place to look for such features is the official documentation for the Android API. *NOTE:* The emulator does not provide feedback for determining whether the vibrate functionality is working correctly. If you test it using the emulator, it is up to you to make sure it is behaving correctly. If the application crashes, make sure that you gave it permission to use the vibrate features of the Android API. You will need a "`uses-permission`" for `android.permission.VIBRATE`

Once you have the vibrate feature working, commit your changes with an appropriate message and push them to your repository.

Tagging

Tag the contents of your Git repository with the name AS2_Complete. Use the menu item VCS→Git→Tag Files and enter the tag name AS2_Complete as well as a useful message.

Push your local repository to the remote GitLab repository one more time (VCS→Git→Push). In GitLab, notice that the tags are not pushed to the remote repository by default. You *can* push tags from the command line interface using `git push origin <tag name>`

Examining The History

Now take a look at the repository history both locally and remotely. You can view the local history of any file by right clicking on it and choosing Git→Show History. You can see the history of the entire project by using VCS→Browse VCS Repository. . . →Show Git Repository Log. . . .

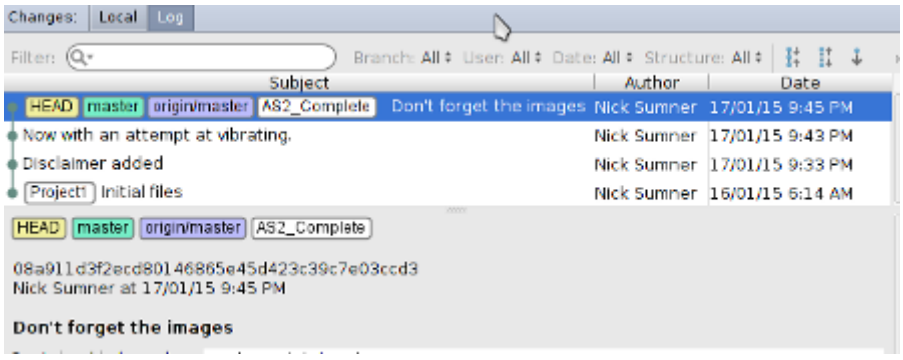

Figure 4: Local Git history in Android Studio

View the history of the entire project and take a screenshot, making sure to show your commit messages, tags, and time stamps. Save this screenshot as

AS2_LocalHistory.png in the docs directory. You can even add it to your repository.

In GitLab, click on the Commits tag and take a screenshot, making sure to capture the dates and commit messages of your commits. Save this screenshot as AS2_RemoteHistory.png in the docs directory.

Deliverables

Submit a ZIP file of your Android Studio project through CourSys.

Locate your project directory in the file system. In Android Studio, you can find the path to your project by right clicking on your project name in the path bar:

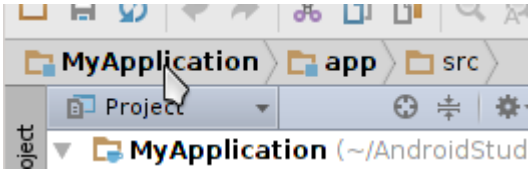

Figure 5: Project name in the path bar

and selecting "Copy Path".

Create a ZIP file containing this directory. This ZIP file should contain not only your code and `.xml` files, but also the `docs/` folder including your new screenshots. If your screen shots in the docs directory are also committed to your local Git repository, then you can use Git in the project directory to create an archive or the repository using:

```
git archive master --format zip --output /path/to/archive.zip
```

If you use this, double check that the archive contains your project directory and screen shots. In Linux, this can also be done with the command:

```
zip -r project2 ProjectName/
```

This will create `project2.zip` containing the directory `ProjectName` and all of its subdirectories.

Please remember that all submissions will automatically be compared for unexplainable similar submissions. Everyone's submissions will be quite similar, given the nature of this assignment, but please make sure you do your own original work; we will still be checking.