Git + Github Exercises

Git vs Github

Git is a distributed revision control system.

GitHub is a web-based Git repository hosting service. (BitBucket is another web site that offers git hosting.) GitHub offers both public free accounts and paid plans for private repositories. (I believe they also offer a few free private repositories to users with .edu email addresses.)

GitHub takes version control a step further by facilitating **social coding**. GitHub makes it easy to contribute to other projects or accept contributes to a project you started. GitHub is located on the web at github.com.

If you don't already have an account at github, create one now. The free account option is sufficient for this exercise.

If you use Windows, you probably want to install a git client on your local machine. Github for Windows (http://windows.github.com) is a popular choice but there are other options including command line clients for Mac, Windows and Linux.

There are GUI clients for using git, but the advantage of using the command line is:

1. It doesn't change. Different GUI clients fall in and out of favor and how a GUI client works can change from one version to another.
2. All major GUI clients also offer a command line option.
3. To debug a problem from a GUI interface, it helps to know the git command being issued.

Exercise 0 – Putting an Android Studio Project on GitHub

This exercise will show you how to put a local Android Studio Project on GitHub.

Step 1. Create a repository on GitHub with the name GPSExample. I recommend adding a Readme file and .gitignore for Android.

Step 3. Use git bash or other git client to clone repository just created:

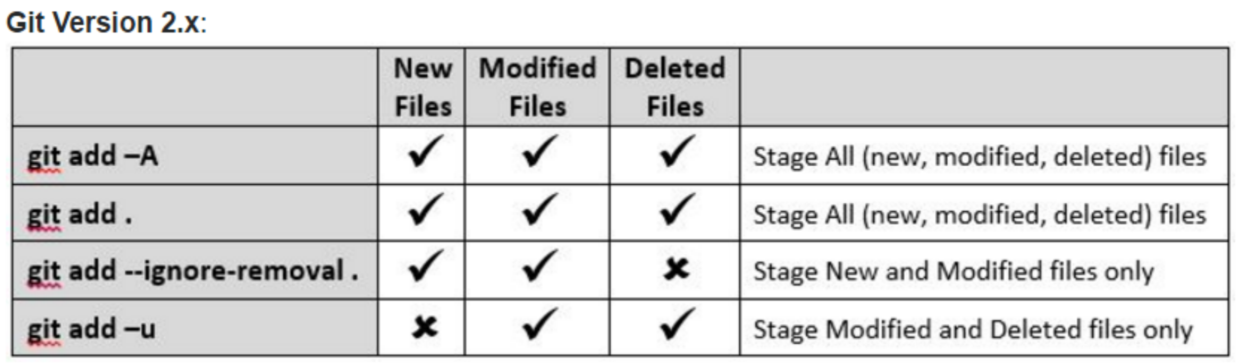
```
git clone https://github.com/<your account>/GPSExample.git
```

Step 4. Copy some existing files to the GPSExample directory you just cloned .

Step 5. Go to the directory. Add, commit and push your changes:

```
git add -A
git commit -m "short description of change"
git push
```

[git add . and git add –A are equivalent in git version 2:



Git Version 2.x:

	New Files	Modified Files	Deleted Files	
git add –A	✓	✓	✓	Stage All (new, modified, deleted) files
git add .	✓	✓	✓	Stage All (new, modified, deleted) files
git add --ignore-removal .	✓	✓	✗	Stage New and Modified files only
git add –u	✗	✓	✓	Stage Modified and Deleted files only

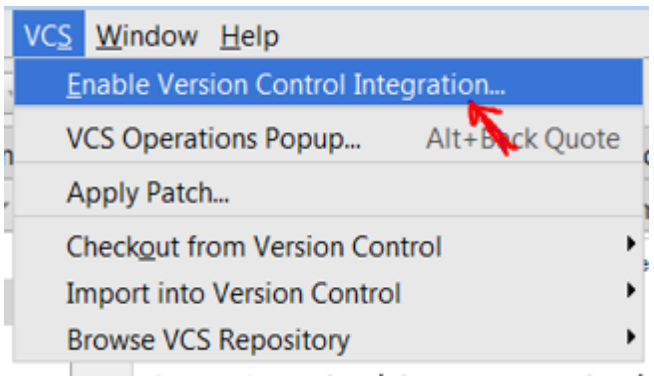
]

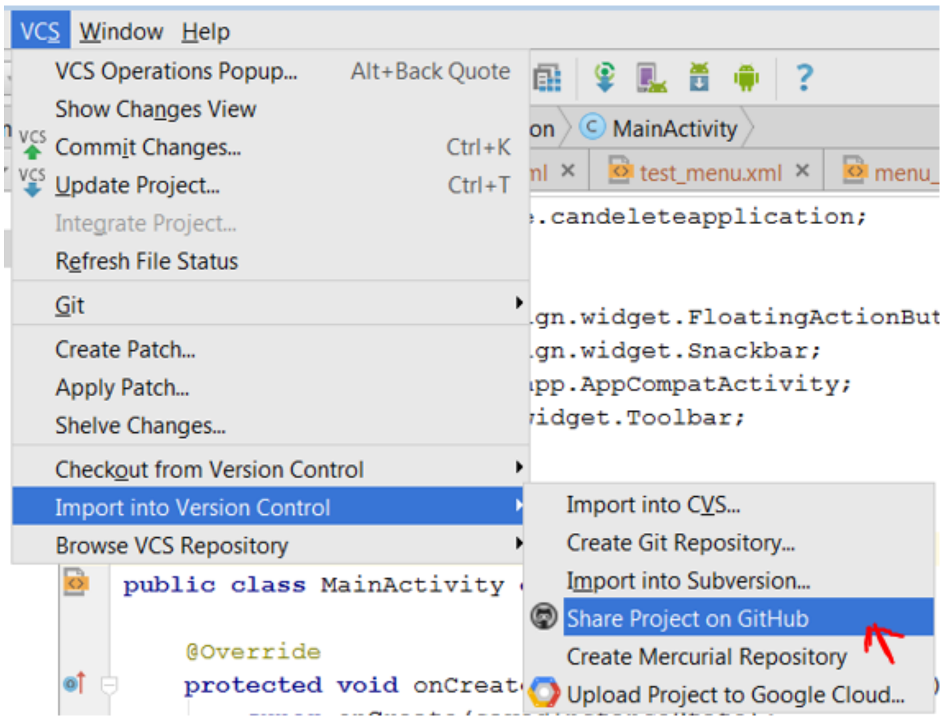
After every major change, repeat step 5.

If you cloned the repository to another computer, execute the following to pull any updates:

```
git pull
```

Or, you can set up a git repository through the GUI:

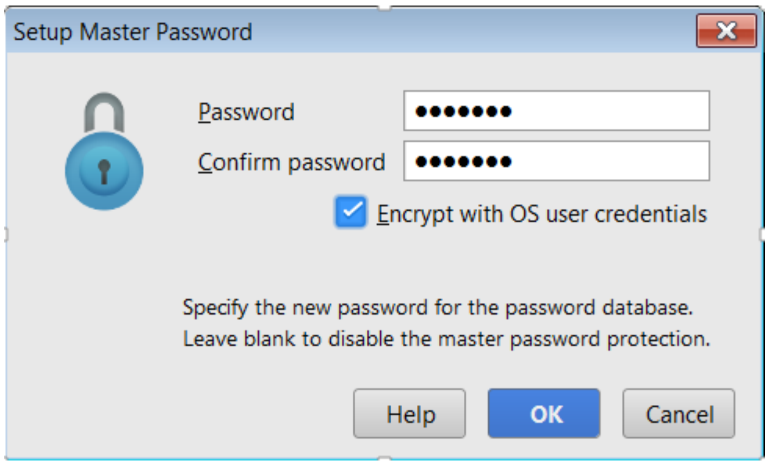

VCS
Window
Help
Enable Version Control Integration...
VCS Operations Popup... Alt+Back Quote
Apply Patch...
Checkout from Version Control
Import into Version Control
Browse VCS Repository



```
.candeleteapplication;

.gn.widget.FloatingActionBut
.gn.widget.Snackbar;
pp.AppCompatActivity;
idget.Toolbar;
public class MainActivity

    @Override
    protected void onCreat
```

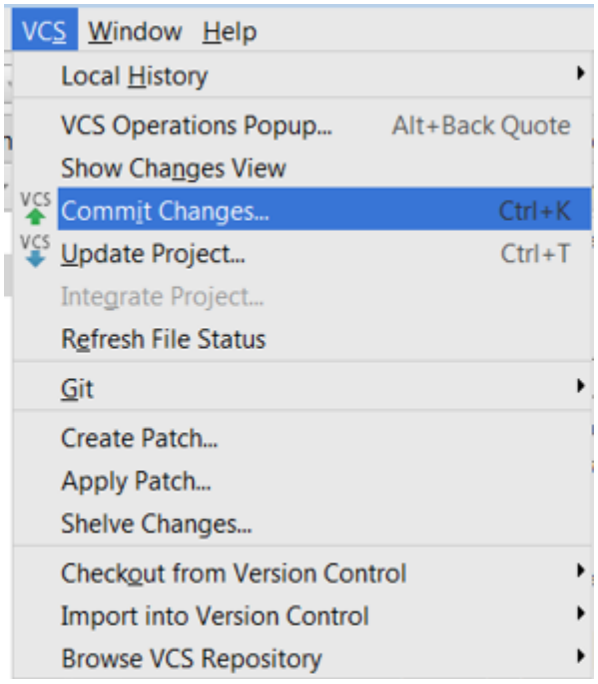


Password: ●●●●●●●

Confirm password: ●●●●●●●

[✓] Encrypt with OS user credentials

Specify the new password for the password database.
Leave blank to disable the master password protection.



Exercise 1 – Creating and using a local repository

This exercise will give you practice with:

- Creating and using a local repository
- Retrieving old versions of a file.
- Using a .gitignore file to keep certain files from being tracked
- Creating a repository on github.com
- Committing and syncing (or pushing) changes to github

Open one of the git shells (command line tools) you downloaded above. I recommend the git bash shell.

Create a directory somewhere on your file system for practice and make this the current directory in the command shell. It's OK if there are existing files in the directory.

Show your current directory (pwd = print working directory):

```
$ pwd
```

Create a directory:

```
$ mkdir gitpractice
```

```
$cd gitpractice
```

Start tracking files and subfolders with git:

```
$ git init
```

Run git status to see current status of the repository:

```
$ git status
```

The first time you use the shell, it is a good idea to set up your identify for committing changes.

To set your identity, enter:

$ git config --global user.name "Your Name"

$ git config --global user.email you@example.com

Check values you just set:

$ git config --list

You may also want to change the editor. I believe the default is vi. To change the editor to notepad, enter:

$ git config --global core.editor notepad

Add a new file to the directory. For example, use notepad to create a file:

$ notepad hello.txt

And add the contents:

```
// Hello world
void main() {
    print "Hello World";
}
```

Run git status:

$ git status

You should see the file as “untracked”.

Add the file with:

$ git add hello.txt

When there are new or changed files in your directory, you have to add them before they will be included in a commit. This is a nice feature because it prevents generated binary files or other new files you don’t want to be tracked from being included in a commit.

Run git status to verify the file was staged:

$ git status

Note, you can use a regular expression to add more than one file. Example: git add *.c

The add command takes the name of a file or directory. If a directory is specified all the contents of the directory and subdirectories are added. For example:

$ git add src

To commit all changes that have been added, enter:

$ git commit –m ‘description of change’

To verify the commit was successful, run git status:

$ git status

Make a change to the file using notepad or other editor.

Run git status:

```
$ git status
```

You should see the status of the file as changed but not staged.

Stage it with:

```
$ git add hello.txt
```

Run git status:

```
$ git status
```

You should see the status of the file as staged but not committed.

Now, commit the changes with:

```
$ git commit –m ‘description of change’
```

Note, git add stages changes to be committed. When you commit you don’t commit the current version of the file, you commit the state of the file at the point when it was lasted added. For example, if you do something like:

Change 1 → add → change 2 → commit

Only change 1 will be committed. Try it. Before you commit, do a git status and you will notice the same file name listed as staged but not committed and as changed but not staged.

Git doesn't require commit messages to follow any specific formatting constraints, but the canonical format is to summarize the entire commit on the first line in less than 50 characters, leave a blank line, then a detailed explanation of what’s been changed. For example:

```
Make greeting time dependent
- print Good Day if hour is 3am – 5pm
- print Good Evening if hour is 6pm – 2am
```

The best way to enter a multi-line comment, is to leave the –m option off the commit. The following will open the default editor for the commit message:

```
$ git commit
```

To see a history of changes:

```
$ git log
```

To see an abbreviated history:

```
$ git log –oneline
```

If there is more than one page, press space bar to advance to the next page or q to quit.

You can also, limit the number of previous commits to show. The following will show the last 3 commits:

```
$ git log –n 3
```

To show the differences introduced in each commit, use the –p option:

$ git log –p

You can of course, combine the two:

$ git log –p –n 3

There are other options for showing magnitude of changes with each commit, author, etc. (See: https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-log)

The 40-character string after commit is an SHA-1 checksum of the commit's contents. This serves two purposes. First, it ensures the integrity of the commit—if it was ever corrupted, the commit would generate a different checksum. Second, it serves as a unique ID for the commit.

To get back to a previous state:

$ git checkout <hash from log command or tag>

The hash can be the full hash value from git log or the abbreviated one from git log –oneline.

Try checking out a previous commit. Browse the file to verify it is an older previous version.

To get back to the latest:

$ git checkout master

You can also checkout a single file from a previous commit:

$ git checkout <hash from log command or tag> filename

To get back to the latest version of the file, enter:

$ git checkout HEAD filename

Sometimes there are files in your directory you don't want tracked by git. For example, in most cases there is no need to track files created by the compiler (e.g. .exe, .class, etc.).

git won't track these files unless you add them, but they will show up as untracked when using the status command. If there are files you know will never be tracked, you can tell git to completely ignore them.

If you create a file in your repository named .gitignore, Git uses it to determine which files and directories to ignore.

A .gitignore file should be committed into your repository (add followed by commit), in order to share the ignore rules with any other users that clone the repository.

Use notepad to create a .gitignore file with the following contents:

```
// Ignore .class files and everything
//   in the bin directory
*.class
bin/
```

Add and commit the .gitignore file to your project:

$ git add .gitignore

```
$ git commit –m ‘created .gitignore’
```

Now, create a bin directory, add a file to it and create a .class file:

```
$ mkdir bin
```

```
$ touch bin/somefile.exe
```

```
$ touch file.class
```

Now, check git status and notice these files are ignored (git doesn’t flag these new files as untracked):

```
$ git status
```

Viewing staged and unstaged changes. git status gives a high-level picture of what files have changed and need to be added and/or committed. For more details on what has changed, you need git diff. git diff will tell you (1) what is changed but not staged, and (2) what is staged and about to be committed. Unlike git status, git diff shows exact lines changed/added/removed.

The following shows changed but not added:

```
$ git diff
```

The following shows exactly what will be added in the next commit (exact lines of each file):

```
$ git diff –staged
```

To delete a file from your project and remove it from being tracked, you can’t just delete and commit. To delete a file enter:

```
$ git rm <filename>
```

```
$ git commit –m ‘message’
```

git rm <filename> removes the file from the directory and causes git to stop tracking it. If you just want git to stop tracking it, use:

```
$ git rm --cached <filename>
```

If you want to rename a file, one option is to rename it outside of git and then delete the old one and add the new one in git:

```
$ mv oldfile newfile
```

```
$ git rm oldfile
```

```
$ git add newfile
```

Or, you can accomplish the same using the git mv command:

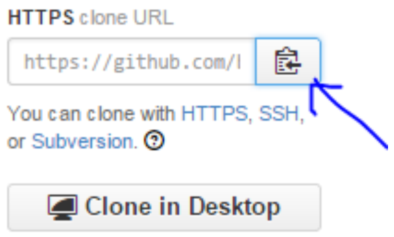
```
$ git mv oldfile newfile
```

Exercise 2 – Cloning an existing repository

Oftentimes the git repository will already exist on a remote server. The common scenario is you join a project and want to modify existing documents or upload new ones. In cases like this, you (1) create a local copy or clone of the repository, (2) add/modify existing documents. (3) push your changes back to the central or upstream repository.

In this exercise, you will clone a repository, make a change to an existing file and upload or push your changes back to the server.

Since you don't have the permission to work on the existing repository directly, you need to **fork** the following repository: https://github.com/tingting703/GPSExample.git. (See Exercise 3's instruction on how to fork a repository.)



Open one of the git shells (command line tools) you downloaded above. I recommend the git bash shell.

Get the URL to the remote repository you want to clone.

The clone command will create a new directory for the cloned project. cd to the directory where you want to create the new repository. Clone the remote repository with:

```
$ git clone https://github.com/<Your_Account_Name>/GPSExample.git
```

The command above will create a new subdirectory GPSExample and initialize it with the remote repository. cd (change directory) into the new directory:

```
$ cd GPSExample
```

Verify it the clone command created remote references to the remote repository:

```
$ git remote –v
```

Modify one of the downloaded files or add a new file to the directory:

```
$ notepad <filename>
```

```
$ echo "new file contents" > newfile.txt
```

```
$ git add <filename>
```

```
$ git add newfile.txt
```

```
$ git commit –m 'practice changes'
```

To push your changes to the remote server:

```
$ git push
```

Browse github.com to verify new changes were uploaded correctly.

When you clone a repository with git clone, it automatically creates a remote connection called origin pointing back to the cloned repository. You can verify this with the following command:

```
$ git remote -v
```

Pro tip! As time goes on others may push their own changes to the central repository. On a regular basis you should update your local copy with changes made on the central repository:

```
$ git pull origin master
```

Exercise 3 – Contributing to an existing github project

Reference for this section: http://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project

This exercise will give you practice with contributing to an existing github.com project. There are two options for contributing to an existing github project. First, the owner of the project could give you read/write access (not very likely), or you could use the fork & pull request method. This exercise uses the fork & pull request method.

In this exercise you will fork a repository, commit and sync changes to your forked copy on github and then issuing a pull request to the owner of the original project to have your changes merged with the original.

First, fork the following repository: https://github.com/tingting703/gitpractice.git. You will find it under the user ID: tingting703.

(Hint: log on to github.com, find the repository, press “Fork”.)

When you fork a github repository, a copy of the repository appears under your account at github.

Clone the forked repository to a local computer. (See exercise #2 above.) (Note, you will have to change tingting703 to your github user ID.)

```
$ git clone https://github.com/<your user ID>/gitpractice.git
```

```
$ cd gitpractice
```

Create a topic branch for performing work.

```
$ git checkout -b EddiesContribution
```

(Or:

```
$ git branch EddiesContribution
$ git checkout EddiesContribution
```

)

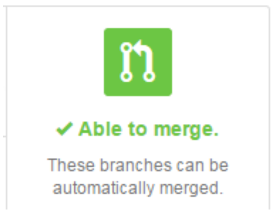
Open the local file References.txt and add a reference. Be sure to include your name so we know who to credit.

Once you are done with all edits and commits, commit your changes to your topic branch.

You could do a git push origin EddiesContribution at this point:

```
$ git push origin EddiesContribution
```

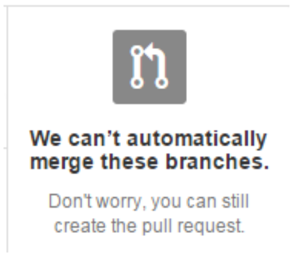
If the repository you forked hasn't changed since you forked it or changes can be automatically merged, you will be able to issue a pull request that can be merged automatically:



✔ Able to merge.

These branches can be automatically merged.

However, if changes were made to the forked repository that conflict with changes you made, you will get a message like the following when you try and issue a pull request:



We can't automatically merge these branches.

Don't worry, you can still create the pull request.

Either way, it is good practice to pull in any upstream changes before issuing a pull request.

To make it easier for the owner to merge your changes, you should add an upstream remote and pull in any upstream changes there are.

The following command will add an upstream remote:

```
$ git remote add upstream https://github.com/tingting703/gitpractice.git
```

Now, pull in upstream changes:

Option 1:

```
$ git fetch upstream
$ git merge upstream/master
```

(Note, you may want to use the --no-ff flag. It will always generate a merge commit: git merge --no-ff upstream/master.)

Option 2:
$ git pull upstream master

If there are no upstream changes since you forked, there will be nothing to merge.

If there are upstream changes but they don't overlap with the changes you made, the changes will be merged automatically.

If there are upstream changes that can't be merged automatically, you will have to manually merge the changes. If there are merge conflicts, you will get a message something to the effect "Automatic Merge Failed". To see which files need to be manually merged, use the git status command:

$ git status

Look for the section "Unmerged paths".

To manually, resolve the conflict, open the file or files with the merge conflicts and look for lines of the form:

```
<<<<<<< HEAD
<Changes you made>
=======
<Changes on the server that conflict with your changes
>>>>>>> 7bcfe2880234048e64254dbc35da1176ecc6f839
```

Manually resolve the conflict. Be sure to remove the surrounding << .. >>> lines. Once done editing, commit changes:

$ git add <filename>

Staging the file marks it as resolved in Git.

$ git commit –m 'resolved conflicts'

$ git push origin EddiesContribution

The above will fail if your topic branch is behind the current contents of the remote repository.

~~Now, send the original author of the repository you forked a pull request.~~

~~(Hint: go to the repository on github.com, click on pull request, click on create pull request, click on send pull request)~~

~~(To accept the pull request, the author will click on "Pull Requests", open the pull request, review it and then click on "Merge Pull Request".)~~