

# System Analysis and Design Lab

## Activity 3.1: Version Control with Git

### Overview

Version Control systems allow developers to work together in an efficient manner. Using these systems, developers work on their changes to the software and can then merge in changes made by others.

- Purpose:** Learn some of the basics for working with the Git version control system.
- Task:** Use the instructions on the following pages create a copy of a repository and then make changes as a part of a group of developers.
- Time:** This task should be completed in your lab class and submitted for feedback before the start of next week.
- Resources:** 1. [Github Help](#)
- Feedback:**
- Next:** When complete, move on to Activity 4.1

### ***Activity 3.1 — Submission Details***

You must submit the following things in the report:

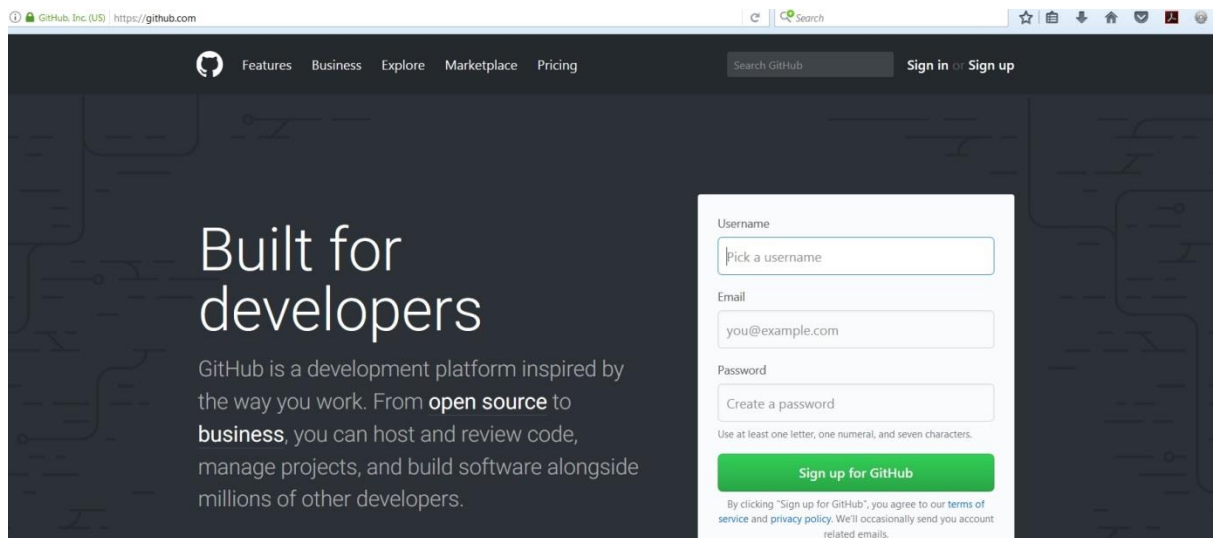
- Images of the git contributions and branches.
- Git reference sheet
- An overview of the process for using Git for revision control systems.

## Instructions

In this task you will work as a team of three developers to create a small two-player card game using SwinGame and C#. The code isn't the focus, so you will be given the changes you need to make. You want to focus on the **process** of using Git to work together as a team.

**Note:** Make sure you have [Mono](#) installed, this will be needed by the project. You can also install [Mono Develop](#).

1. Navigate to the [GitHub](#) website.
2. Sign up for an account, or login if you already have one account.



3. Make sure that you have Git setup on your machine — See [Setup Git](#) on GitHub

**Note:** This will already be setup on the lab machines.

4. Install a git command line tool installed on your machine. You can download installers for Linux, Mac and Windows at <http://git-scm.com/downloads>.
5. Test your setup. Open a Terminal window and type **git --version** you should see something like this:

A screenshot of a terminal window titled "MINGW64:/h/". The prompt is "manwar@EN404-18-MANWAR MINGW64 ~". The user has entered the command "\$ git --version" and the output is "git version 2.16.2.windows.1". The prompt is now "\$".

```
manwar@EN404-18-MANWAR MINGW64 ~
$ git --version
git version 2.16.2.windows.1
manwar@EN404-18-MANWAR MINGW64 ~
$
```

6. Setup your name by running the following command from the Terminal:

```
git config --global user.name "YOUR NAME"
```

7. Use the following command to setup your email:

```
git config --global user.email "YOUR EMAIL ADDRESS"
```

8. Optionally setup a password cache. A credential cache will keep track of your passwords so that you don't need to enter your passwords all the time when saving your work to the server. See [Caching your GitHub Password in Git](#).

These settings will be used when you are interacting with git. You want to make sure that these are correct, and the email matches the email you used when you created your account on GitHub. You now have git setup on your machine! So now you just need a team to work with...

9. Get into a team with two other developers — each team should have 3 members!

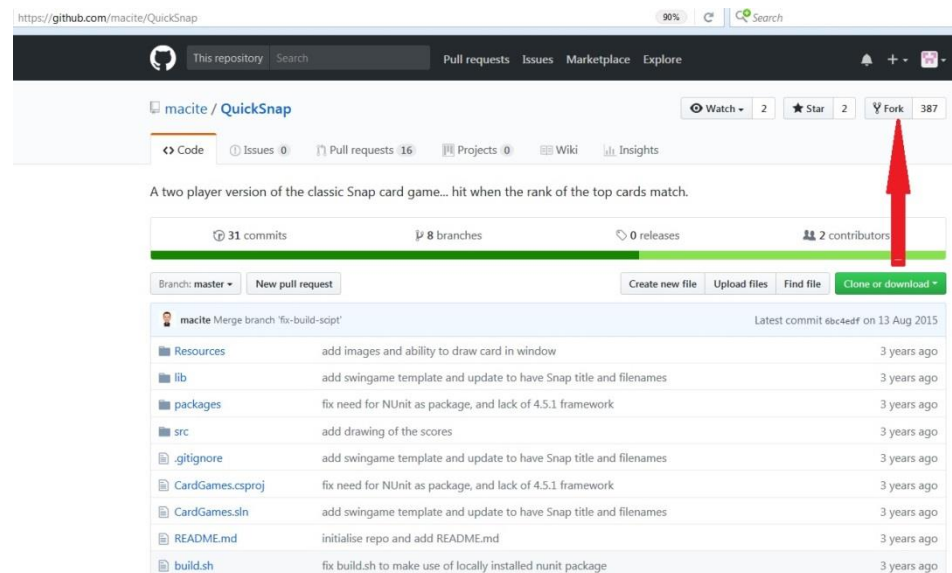
**Note:** If needed one team member can perform multiple roles. Check with your tutor about how to do this.

10. Assign roles to each of the three group members.
  - **Team Leader** (TL) — will setup the project and add fellow collaborators to the team.
  - **UI Coder** (UC) — will focus on making changes to the graphical part of the game.
  - **Game Developer** (GD) — will develop aspects of the backend.

Now that you are in your team, you can work through the following process. Identify the steps related to your role in the project. You will need to gather screenshots and take notes at each stage.

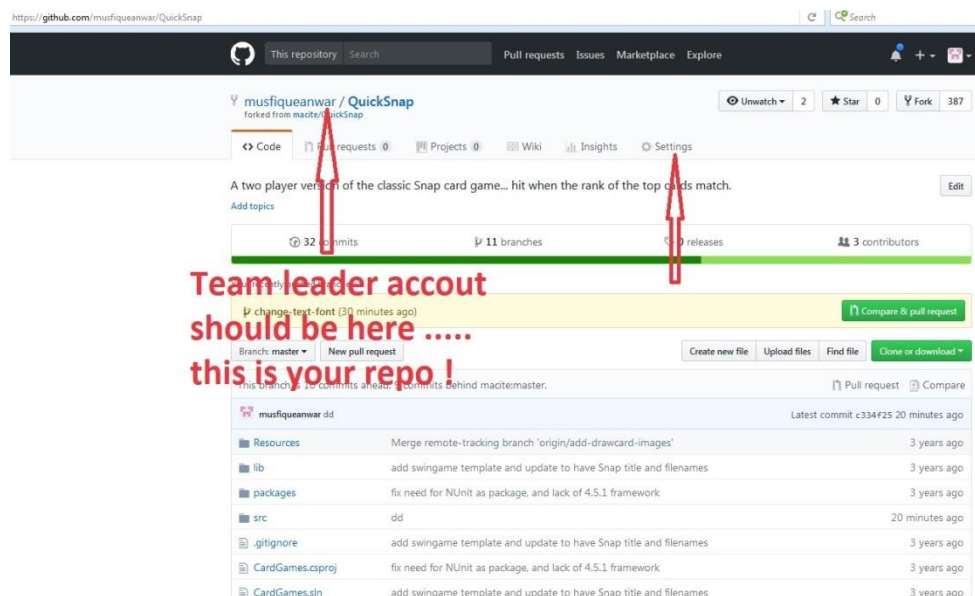
As a team get together at the Team Leader's machine. The first task is to get a copy of the project and ensure that you are all setup as collaborators!

11. TL: Navigate to <https://github.com/macite/QuickSnap> — this is the start of the project.
12. TL: Fork the repository — click the **Fork** button.

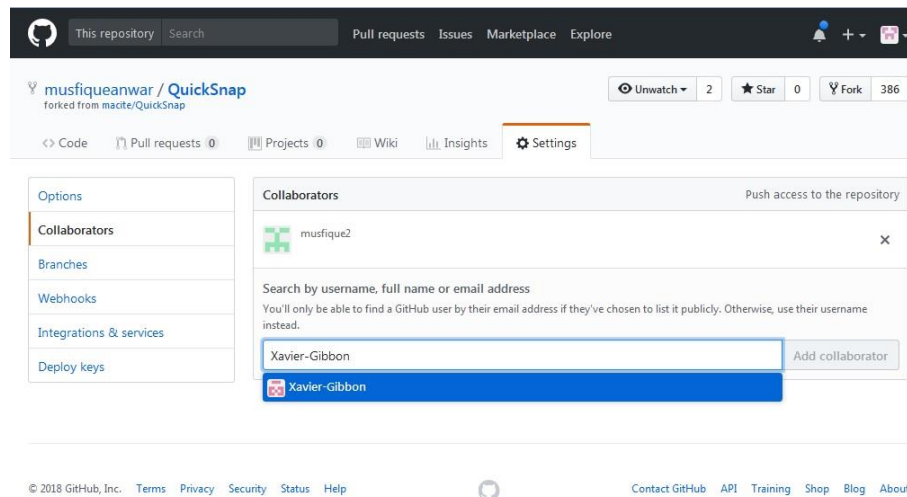


**Note:** Repository is main term used for a project in a version control system. When you fork a repository, you get a copy of that project and its entire history!

13. TL, UC, GD: Notice that fork is now a part of the Team Leader's account. The next step will be to add the UI Coder and Game Developer accounts as collaborators.
14. TL: Click the **Settings** to get to the Repository's settings.



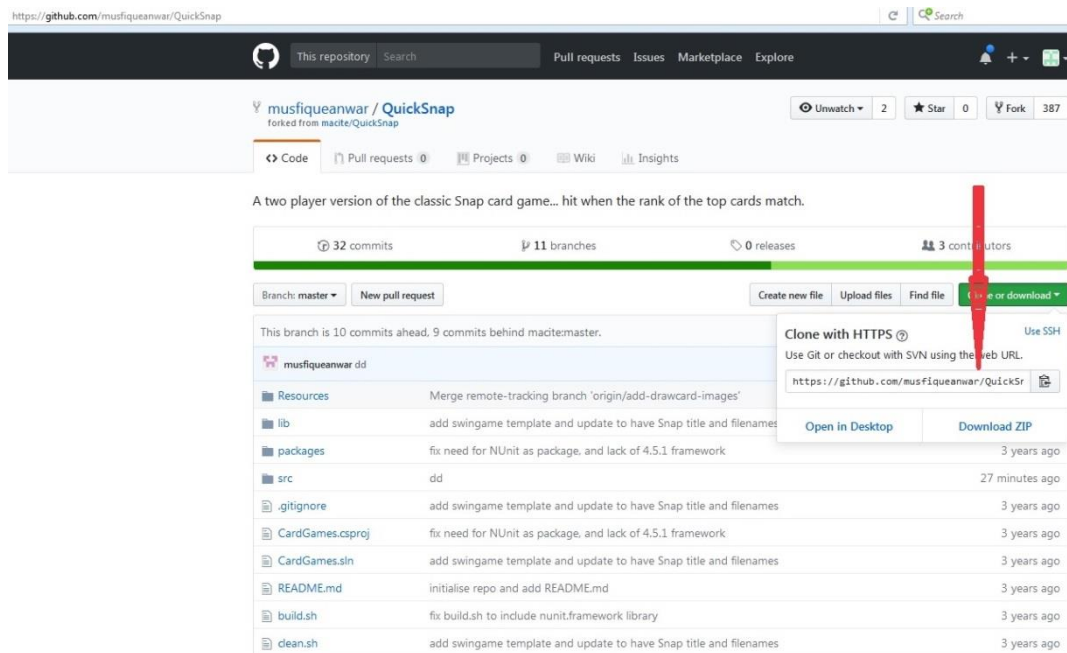
15. **TL**: Select the **Collaborators** tab and enter the names of your two team members. Add them both as collaborators.



16. **UC, GD**: Return to your own computer.
17. **UC, GD**: Once the Team Leader has added you to the project, refresh your GitHub page in the browser and make sure you can see the TL's fork of the QuickSnap repo.

Now that you are all in the repository as collaborators, you can start to work together on this code.

With git you need to get your own local copy of the repository. You then work on that repository and push your changes back to the server when you have something cool working that you want to share with the other team members or when you want to make sure your work is backed up!

18. Copy the projects **HTTPS clone url** from GitHub.

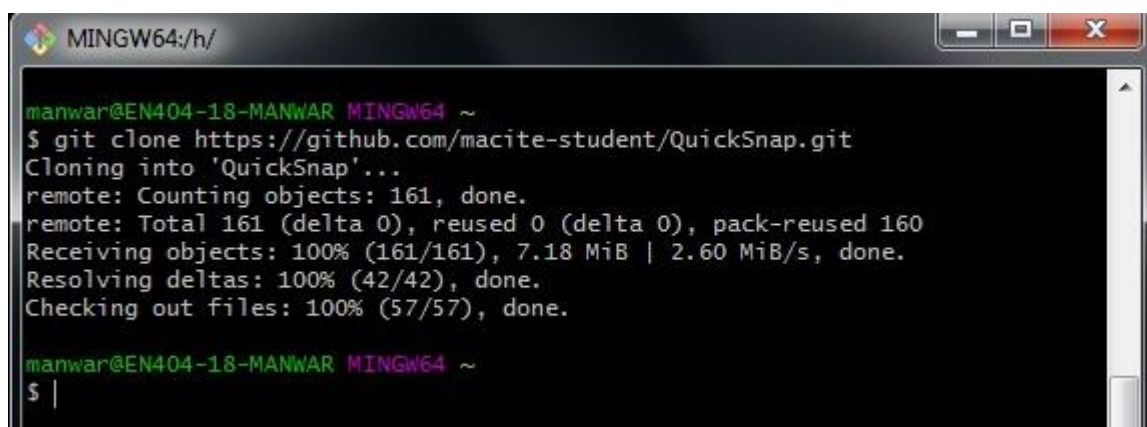
## 19. Switch to your Terminal.

## 20. Navigate to a directory where you want to store the project's code. For example, /c/ Users/username/Documents/Code or something similar.

**Note:** When you get the project it will go into a subdirectory based on the project's name. So don't create a directory for the project itself at this stage.

## 21. At the terminal type the following command, pasting in the URL (example is shown).

```
git clone https://github.com/macite-student/QuickSnap.git
```

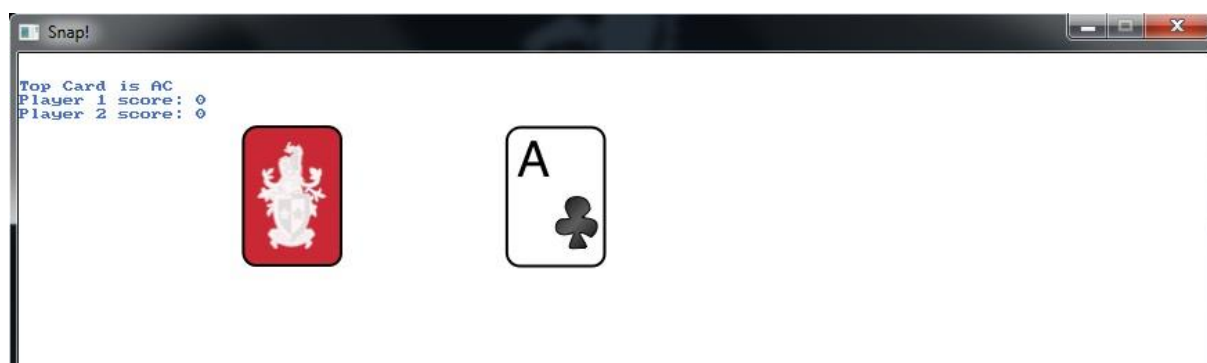


You should now be able to see all of the project's files in the **QuickSnap** folder. This is your working copy of the projects.

22. Test that the project works currently. In your terminal window you should be able to compile and run the project with the following steps.

```
cd QuickSnap
./build.sh
./run.sh
```

At this stage you should be able to flip through the cards in the Deck by pressing the Space-bar. Each card should be drawn to the screen in both Text and an Image.



There are the following issues with the program:

- Things for the **UI Coder (UC)** to fix:
  - The text for the score should be drawn using a more interesting font.
- Things for the **Game Developer (GD)** to fix:
  - Pressing space should start the game, with it automatically flipping through the cards.
  - The cards do not get Shuffled.
- Things for the **Team Leader (TL)** to fix:
  - Pressing Left Shift should hit for player 0, and Right Shift should hit for player 1.

You can all work on this project now, and get git to merge all of your changes!



**Note:** In this task you will work one at a time, but these tasks could actually be performed in parallel! Let's keep it simple to start with.

### UI Coder Tasks:

Get your team together at your computer to observe the process.

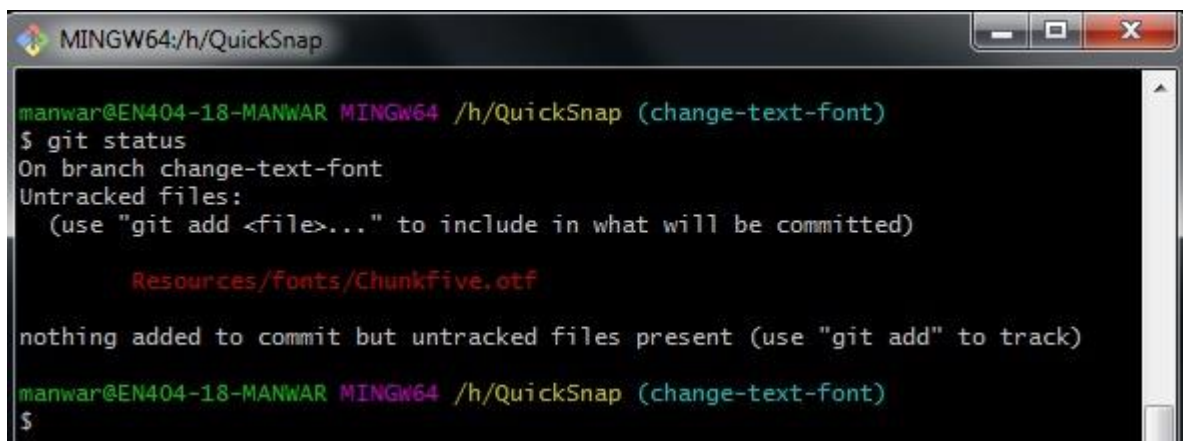
23. In the repository you should always be working on a **branch** other than the master — which keeps a "good" state of the project. Let's change the text font first.

Start a new branch using the following command:

```
git checkout -b change-text-font
```

24. Get the [chunkfive](#) font. Copy the OTF file into **Resources/fonts** folder of the project.
25. Check the status of the repository, and you should be able to see the new file. The image should be listed as "Untracked" as it is not in the repository.

```
git status
```



```
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)
$ git status
On branch change-text-font
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Resources/Fonts/Chunkfive.otf

nothing added to commit but untracked files present (use "git add" to track)
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)
$
```

26. Add the file to the repository using the following command. This will stage this change, ready for you to commit it when you are ready.

```
git add Resources/fonts/Chunkfive.otf
```

27. Open the **SnapGame.cs** file, which contains the main program instructions. Locate the **LoadResources** method and add a line of code to change the text font. (Only add the highlighted line.)

```
public static void LoadResources ()
{
    Bitmap cards;
    cards = SwinGame.LoadBitmapNamed ("Cards", "Cards.png");
    SwinGame.BitmapSetCellDetails (cards, 82, 110, 13, 5, 53);

    SwinGame.LoadFontNamed ("GameFont", "Chunkfive.otf", 12);
}
```



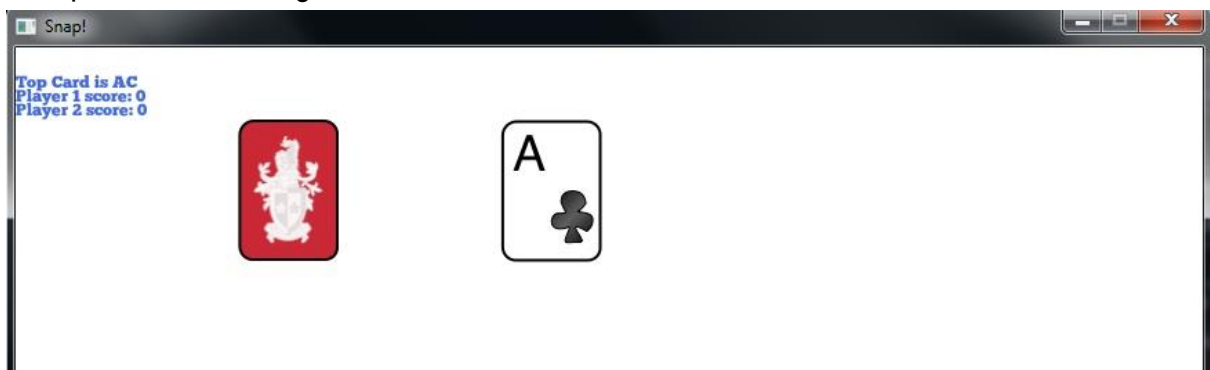
28. Now locate the **DrawGame** method in **SnapGame.cs** file and update the following lines

```
private static void DrawGame(Snap myGame)
{
    SwinGame.ClearScreen(Color.White);

    // Draw the top card
    Card top = myGame.TopCard;
    if (top != null)
    {
        SwinGame.DrawText ("Top Card is " + top.ToString (),
        Color.RoyalBlue, "GameFont", 0, 20);
        SwinGame.DrawText ("Player 1 score: " + myGame.Score(0),
        Color.RoyalBlue, "GameFont", 0, 30);
        SwinGame.DrawText ("Player 2 score: " + myGame.Score(1),
        Color.RoyalBlue, "GameFont", 0, 40);
        SwinGame.DrawCell (SwinGame.BitmapNamed ("Cards"),
        top.CardIndex, 350, 50);
    }
    ...
    ...}

```

29. Compile and run using **./build.sh** and **./run.sh**.



30. Check the status of the project again to see the new changes. Note that the new file is ready to commit, but that the changes to SnapGame are *not staged*.

```

MINGW64/h/QuickSnap
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)
$ git status
On branch change-text-font
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   Resources/fonts/Chunkfive.otf

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   src/SnapGame.cs

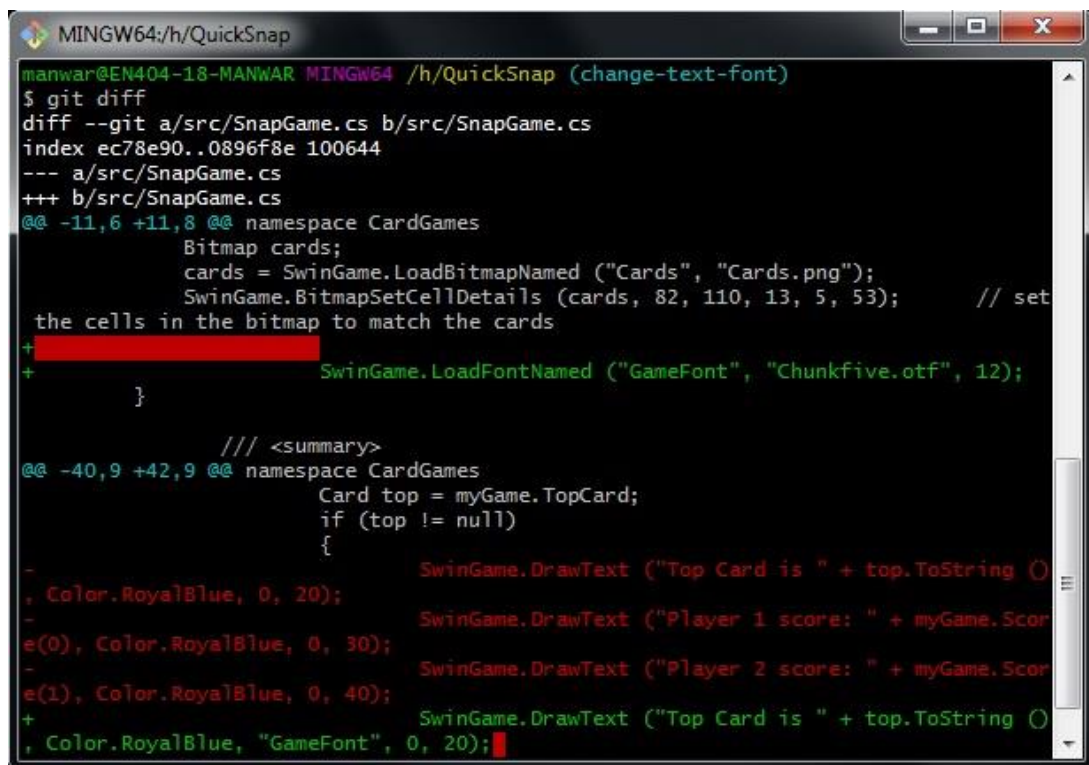
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)
$ |

```

31. To view the changes that will be made use git's diff. You can get the changes of all files in the project or for a single file.

```
git diff
```

```
git diff src/SnapGame.cs
```



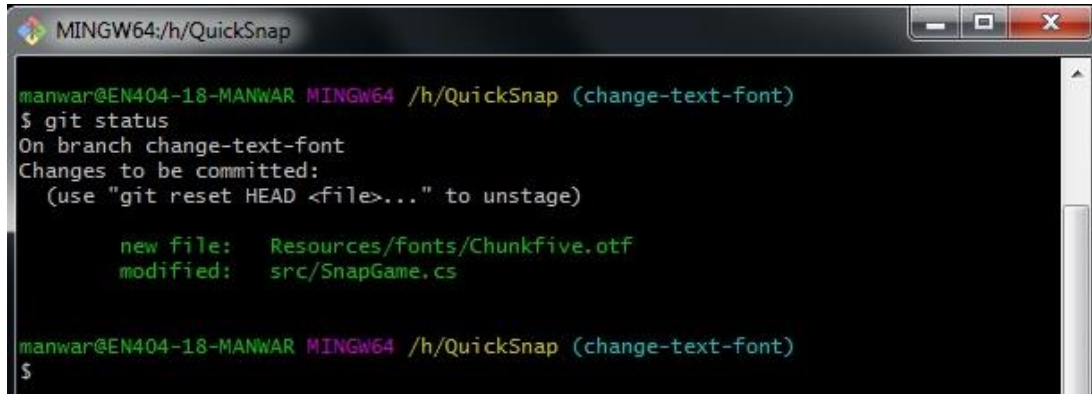
```
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)
$ git diff
diff --git a/src/SnapGame.cs b/src/SnapGame.cs
index ec78e90..0896f8e 100644
--- a/src/SnapGame.cs
+++ b/src/SnapGame.cs
@@ -11,6 +11,8 @@ namespace CardGames
     Bitmap cards;
     cards = SwinGame.LoadBitmapNamed ("Cards", "Cards.png");
     SwinGame.BitmapSetCellDetails (cards, 82, 110, 13, 5, 53);    // set
the cells in the bitmap to match the cards
+
+    SwinGame.LoadFontNamed ("GameFont", "Chunkfive.otf", 12);
+
     }

    /// <summary>
@@ -40,9 +42,9 @@ namespace CardGames
     Card top = myGame.TopCard;
     if (top != null)
     {
-        SwinGame.DrawText ("Top Card is " + top.ToString ()
-        , Color.RoyalBlue, 0, 20);
-        SwinGame.DrawText ("Player 1 score: " + myGame.Score
-        e(0), Color.RoyalBlue, 0, 30);
-        SwinGame.DrawText ("Player 2 score: " + myGame.Score
-        e(1), Color.RoyalBlue, 0, 40);
+        SwinGame.DrawText ("Top Card is " + top.ToString ()
+        , Color.RoyalBlue, "GameFont", 0, 20);
     }
```

32. Press q to close the diff.

33. Add the changes to SnapGame.cs to the repository, ready to be committed. Then check the status to see that everything is added.

```
git add src/SnapGame.cs
git status
```

A screenshot of a terminal window titled 'MINGW64:/h/QuickSnap'. The prompt is 'manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)'. The user has entered '\$ git status'. The output shows 'On branch change-text-font' and 'Changes to be committed: (use "git reset HEAD <file>..." to unstage)'. It lists 'new file: Resources/fonts/Chunkfive.otf' and 'modified: src/SnapGame.cs'. The prompt returns to '\$'.

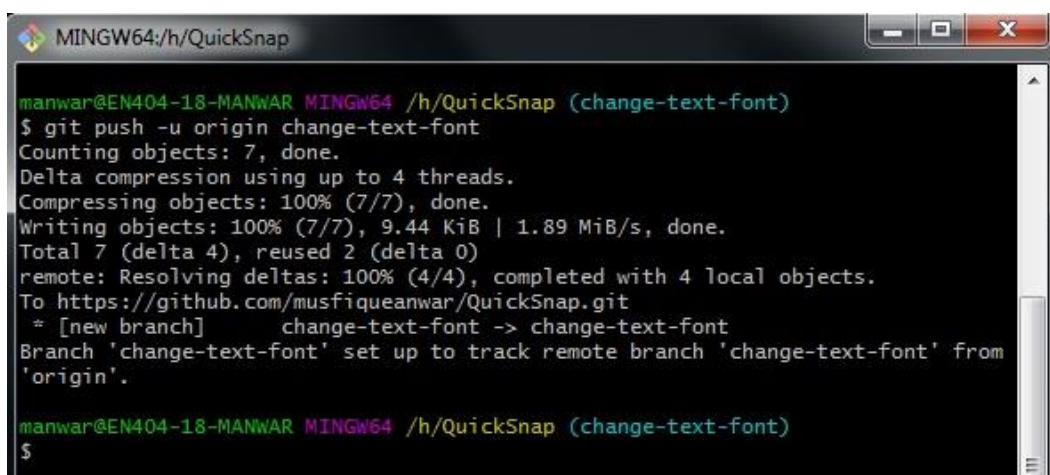
34. Now you need to **commit** the changes to save them into the repository. To do this you will need to provide a message, indicating what the commit does to the repository.

```
git commit -m "add new text font in the game"
```

**Note:** The message is what the commit does, so it will **add** the new font etc. Do not state what you did, so not "added textfont" for example.

35. At this stage your changes are not on the server. You can now **push** your changes so that they are backed up. The first time you push a new branch you must tell git you want it to do up to the server (to the *origin*) and give it a name.

```
git push -u origin change-text-font
```

A screenshot of a terminal window titled 'MINGW64:/h/QuickSnap'. The prompt is 'manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (change-text-font)'. The user has entered '\$ git push -u origin change-text-font'. The output shows progress for counting objects, delta compression, and writing objects. It then shows 'remote: Resolving deltas: 100% (4/4), completed with 4 local objects.' and 'To https://github.com/musfiqueanwar/QuickSnap.git'. It indicates a new branch is being set up to track the remote branch 'change-text-font' from 'origin'. The prompt returns to '\$'.

36. Confirm that your changes are actually on the server by visiting the website.

Now let's switch to the **Game Developer** in your team, and get the cards flipping automatically.

### Game Developer Tasks:

Get your team together at your computer to observe the process.

37. Start by creating a new branch, name it add-automatic-card-flip

```
git checkout -b add-automatic-card-flip
```

38. Open the **Snap.cs** file in the GameLogic folder. It contains the game logic for the two player snap game.

39. Firstly we need to create the Game Timer. Locate then update the constructor to create this object.

```
public Snap ()
{
    _deck = new Deck ();
    _gameTimer = SwinGame.CreateTimer ();
}
```

40. Now locate the Start method, and add a line of code to start the game timer.

```
public void Start()
{
    if ( ! IsStarted )           // only start ...
    {
        _started = true;
        _deck.Shuffle ();        // Return ...

        FlipNextCard ();        // Flip ...
        _gameTimer.Start();
    }
}
```

41. Locate the **Update** method and add an if statement to check how much time has passed. When more than the flip time (1 second by default) the timer will be reset and the next card flipped.

```
public void Update()
{
    if (_gameTimer.Ticks > _flipTime)
    {
        _gameTimer.Reset ();
        FlipNextCard ();
    }
}
```

42. Open the **SnapGame.cs** file, which contains the main program instructions.

43. Locate the **HandleUserInput** method and change (replace old code) pressing the spacebar so that it tells the game to *Start*.

```
private static void HandleUserInput (Snap myGame)
{

```

```
//Fetch the next batch of UI interaction
```

```
SwinGame.ProcessEvents();
```

```
if (SwinGame.KeyTyped (KeyCode.vk_SPACE))
{
```

```
    myGame.Start ();
```

```
}
```

```
}
```

44. Go back to the **Snap.cs** file, and add code to stop the game timer when the player hits the cards. (Only add the highlighted line.)

```
public void PlayerHit (int player)
```

```
{
```

```
    if ( player >= 0 && player < _score.Length && ...)
    { ... }
```

```
    // stop the game...
```

```
    _started = false;
```

```
    _gameTimer.Stop ();
```

```
}
```

45. Switch to the terminal and compile and run the program (`./build.sh` then `./run.sh`).
46. Pressing spacebar should now start the game flipping through the cards on its own!

**Note:** If the right people are doing the right job, the Game Developer should NOT see the new background yet. Why? That is in a different branch! You can work on your features without worrying about what others are doing... for a time.

47. Now check your status to see what has changed.

```
git status
```

48. You can also check the changes in the files using git diff.

```
git diff
```

**Note:** Press the spacebar to move through the pages in diff. Lines in red starting with a - have been removed, those in green with a + have been added. Pressing **q** gets you out of the diff.

49. Things look good, so you can now stage the files using git add, and commit them to the repository. Because you don't have any new files, and you want to add all the changes to the commit, you can do this with the -a option on git commit.

```
git commit -am "add game timer logic to Snap and have
the game started when the spacebar is pressed"
```

**Note:** Watch out, git commit -am "." will not add any new files. So only use this if you have just changed existing files.

50. Push your changes to the server, so that you won't lose them if something happens to your machine.

```
git push -u origin add-automatic-card-flip
```

Now let's switch back to the **Team Leader**, and update the players hitting left and right shift to snap the cards.

### Team Leader Tasks:

Get your team together at your computer to observe the process.

51. Start by creating a branch, add-player-hit

```
git checkout -b add-player-hit
```

52. Open the **SnapGame.cs** file and locate the **HandleUserInput** method.

53. Add the following if statements to check if both, or one of the players has *hit*. When both hit nothing will happen — it was a dead heat. Otherwise, whichever player has hit will update the game.

```
private static void HandleUserInput (Snap myGame)
{
    //Fetch the next batch of UI interaction
    SwinGame.ProcessEvents ();

    if (SwinGame.KeyTyped (KeyCode.vk_SPACE))
    {
        myGame.FlipNextCard ();
    }

    if (myGame.IsStarted)
    {
        if (SwinGame.KeyTyped (KeyCode.vk_LSHIFT) &&
            SwinGame.KeyTyped (KeyCode.vk_RSHIFT))
        {
            //TODO: add sound effects
        }
        else if (SwinGame.KeyTyped (KeyCode.vk_LSHIFT))
        {
            myGame.PlayerHit (0);
        }
        else if (SwinGame.KeyTyped (KeyCode.vk_RSHIFT))
        {

```

```
        myGame.PlayerHit (1);  
    }  
}  
}
```



54. Open the **Snap.cs** file and locate the **PlayerHit** method.
55. Update it so that the player's score is decreased if they don't snap two cards with the same rank.

```
public void PlayerHit (int player)
{
    //TODO: consider deducting score for miss hits???
    if ( player >= 0 && player < _score.Length &&           // ...
        IsStarted &&                                         // ...
        _topCards[0] != null &&
        _topCards[0].Rank == _topCards [1].Rank) // ...
    {
        _score[player]++;
    }
    else if ( player >= 0 && player < _score.Length)
    {
        _score[player]--;
    }

    // stop the game...
    _started = false;
}
```

56. Switch to the terminal and compile and run the program (./build.sh then ./run.sh).
57. Press spacebar to flip the first card, then use left and right shift to reduce the scores of both players.
58. Check the status of the repository, then commit your changes and push to the server.

```
git status
git diff
git commit -am "add player hit with left and right
shift, with score decrement for miss hits"
git push -u origin add-player-hit
```

Now lets switch back to the **Game Developer**, who can start to merge their changes into the master branch.

## Game Developer Tasks:

Get your team together at your computer to observe the process.

You are currently on the add-automatic-card-flip branch, but we have made all of the necessary changes. What we can do now is merge all of our changes back into the master branch.

59. Checkout the master branch. This will switch everything back to what it was like in master when you started.

```
git checkout master
```

**Note:** Checkout -b will create a new branch, without -b and you switch to an existing branch.

60. Pull any changes that others have made. You should always do this before you merge so that you have the latest copy of master.

```
git pull
```

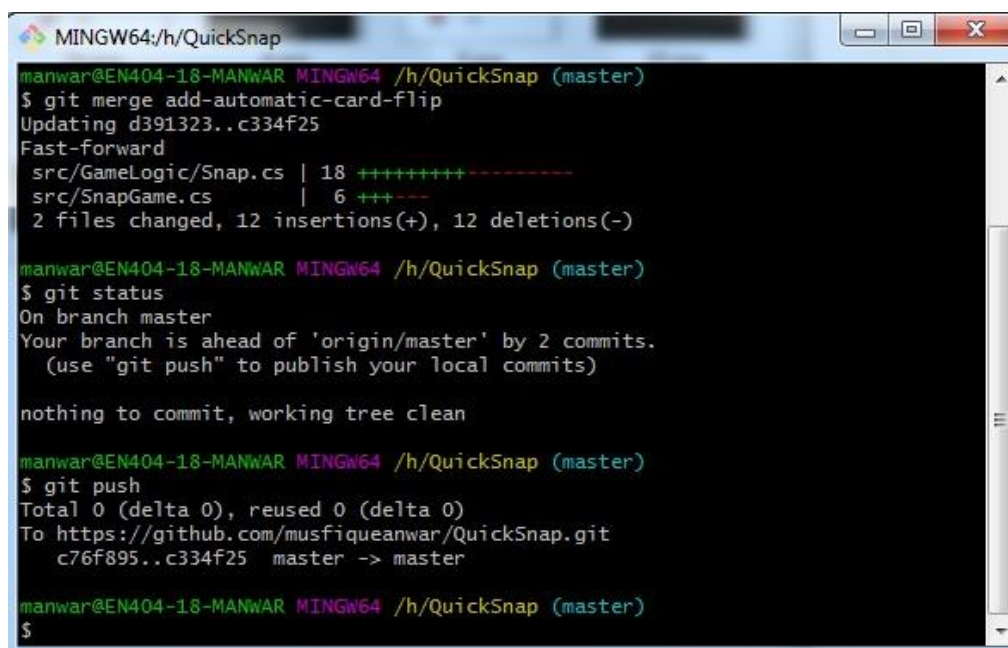
61. Now you can merge your branch into master. This will take all of your changes and apply them to the master branch.

```
git merge add-automatic-card-flip
```

62. When you check the status, notice that we are now ahead of the server's master branch (origin/master). Push your changes so that master is updated for everyone.

```
git status
```

```
git push
```



```
MINGW64:/h/QuickSnap
manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (master)
$ git merge add-automatic-card-flip
Updating d391323..c334f25
Fast-forward
 src/GameLogic/Snap.cs | 18 ++++++-----
 src/SnapGame.cs       |  6 +---
 2 files changed, 12 insertions(+), 12 deletions(-)

manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (master)
$ git push
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/musfiqueanwar/QuickSnap.git
 c76f895..c334f25  master -> master

manwar@EN404-18-MANWAR MINGW64 /h/QuickSnap (master)
$
```

**Note:** No need for -u on this push as the branch exists on the server already!

Lets switch over to the UI Coder again to have them update the game text font.

**UI Coder Tasks:**

63. Switch to the master branch, and **pull** the remove changes.

```
git checkout master  
git pull
```

64. Now merge your branch in ..... notice that git does all of the hard work for you !

```
git merge change-text-font
```

65. Switch to the terminal and compile and run the program (`./build.sh` then `./run.sh`).

66. Push the changes back to the server!

```
git push
```

Notice now that you have your work, and the automatic card flipping! Git takes care of merging as much of the work as it can. In some cases you will need to intervene, but most of the time it should combine (merge) multiple peoples work without issue.

Now lets switch back to the Team Leader to finish off the last changes merge them in.

**Team Leader Tasks:**

Get your team together at your computer to observe the process.

67. Switch to the master branch, and **pull** the remove changes.

```
git checkout master  
git pull
```

68. Merge in your changes.

```
git merge add-player-hit
```

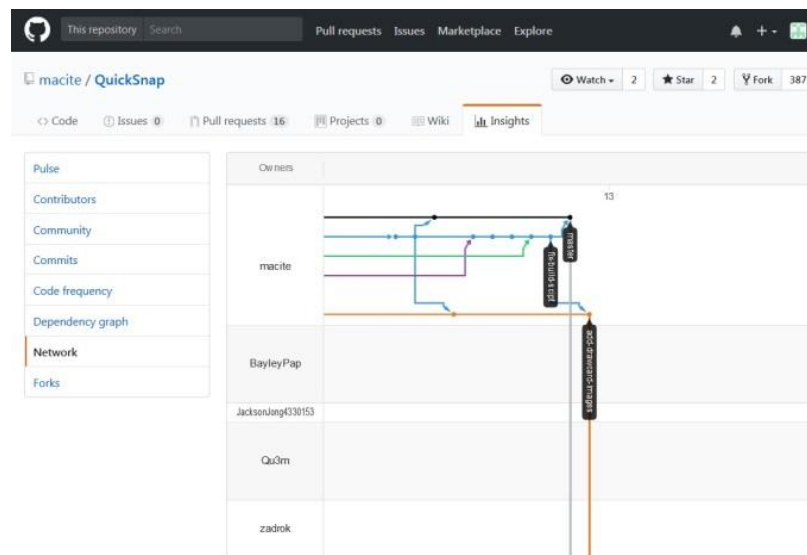
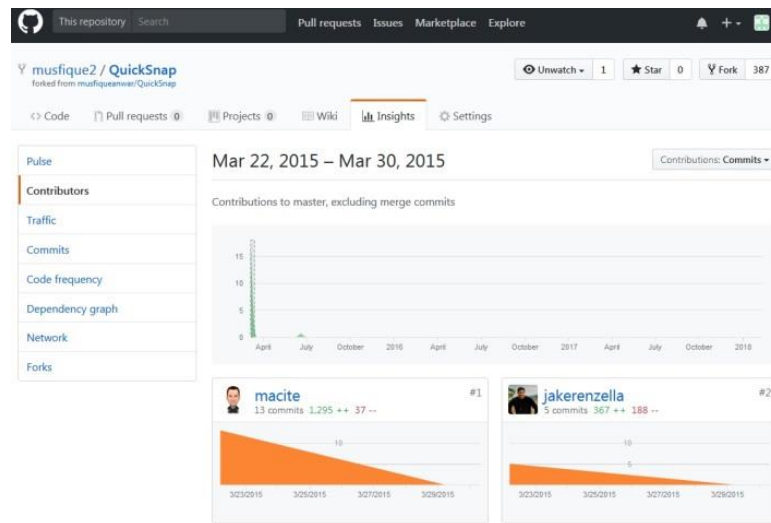
69. Switch to the terminal and compile and run the program (`./build.sh` then `./run.sh`).  
You should be able to see all of the team's changes!

70. Push the changes back to the server!

```
git push
```

## Prepare Submission

On GitHub select the **Graph** tab and get screenshots of the **Contributors** page and the **Network** page.



Individually, prepare a GitHub reference sheet with an outline (list and description) of the commands you have learnt (including the screenshots showing you executing those description) and an overview of the process for using Git for revision control systems. A diagram with numbered steps can help present the overview. Make sure that you include screenshots showing the results of running the game after your modifications.

Individually, submit your screenshots and reference sheet as a single document.