

Introduction to Git & GitHub

Aim

1. To set up and become familiar with the course practices for project releases
2. Get an introduction into how to use git effectively

Before starting your project repository, you must have completed the following tasks as a preparation.

0. Learn Git!

Watch the following videos by Hussein to familiarise yourself with git

[Git videos - Google Drive](#)



1. Sign up for GitHub

GitHub is a version control platform (like Bitbucket or GitLab) that uses git. It is a great way for developers to collaborate with one another. It will be the primary source of starter code distribution as well as where you will submit project source code and reports before being marked by your mentors. You will have to create an account here before you can start. Getting familiar with GitHub and how to use it is the aim of this lab.

Instructions:

- I DON'T have an account on GitHub: Create an account at GitHub using either your student email (zID@student.unsw.edu.au) or your personal email.
- **Make sure to verify your email address. If you do not get an email straight away, go to <https://github.com/settings/emails> and click resend**

Emails

aarthi22@optusnet.com.au	Primary	Private	Notifications	
z5209828@student.unsw.edu.au	Unverified	Verification email sent.	Resend	

- I ALREADY have an account on GitHub: Continue to the next step.

2. SENG3011 GitHub Organisation Membership

A GitHub organisation is a way that people can work together and have group ownership of repositories. You will be added to our organisation where we will release the starter code and solutions.

Instructions:

1. Follow the link and sign in. First with your cse credentials (if the *Authentication required* alert box is shown), then click the **Sign in to GitHub** button and use your GitHub credentials <https://cgi.cse.unsw.edu.au/~se3011/github/run.cgi/login>
2. Go to <https://github.com/orgs/unsw-se3011/invitation> to accept the invitation to join the organisation (invitation may take up to few minutes)
3. Click **Join unsw-se3011** and you should now be part of the organisation :)

3. Add your ssh key

There are two ways to authenticate with GitHub when you are working on your own machine:

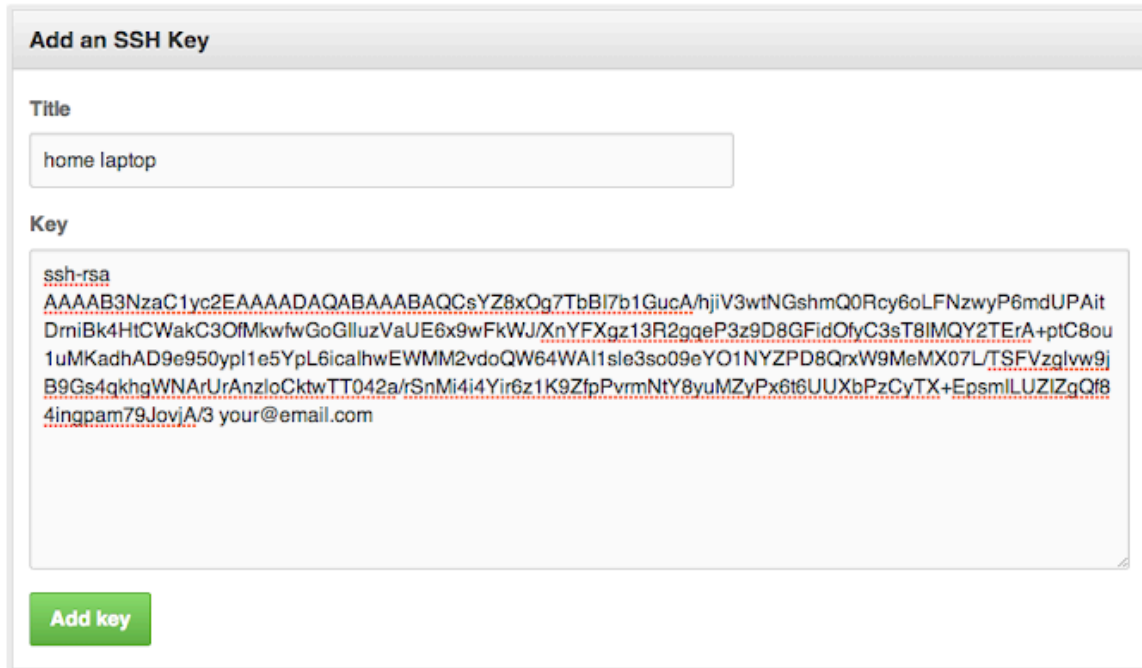
1. Use your username and password every time
2. Authenticate using your SSH key which effectively links a certain machine to GitHub

Option 1 can get quite tedious when working consistently so we will be using the second option. Open up a terminal and run the following commands in terminal (*use the email you have used for GitHub*):

Instructions:

1. `ssh-keygen -t rsa -C "github_email@example.com"`
2. Hit Enter 3 times to accept default location and skip the password creation step. (You can **ignore** the output of this command)
3. `cat ~/.ssh/id_rsa.pub`
4. Copy the entire output of the above command, including the *ssh-rsa* at the start
5. Go to <https://github.com/settings/keys> and click **New SSH Key**

6. Enter *your name* or *your device's name* as the title and paste the key (copied in the above step) into the text field. It should look something like below



Add an SSH Key

Title

home laptop

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQBAQCsYZ8xOg7TbBI7b1GucA/hjiV3wtNGshmq0Rcy6oLFNzwyP6mdUPAit
DmiBk4HtCWakC3OfMkwfwGoGlluzVaUE6x9wFkWJ/XnYFXgz13R2gqeP3z9D8GFidOfyC3sT8IMQY2TErA+ptC8ou
1uMKadhAD9e950ypl1e5YpL6icalhwEWMm2vdoQW64WA11sle3so09eYO1NYZPD8QrxW9MeMX07L/TSFVzqlvw9j
B9Gs4qkqhWNArUrAnzloCktwTT042a/rSnMi4i4Yir6z1K9ZfpPvmNTY8yuMZyPx6t6UUXbPzCyTX+EpsmLUZIZgQf8
4ingpam79JovjA/3 your@email.com
```

Add key

NOTE: You will have to repeat this process if you change machines.

PART 1 - GitHub Exercises (5 marks)

0. Install git

Throughout the course you will need to be comfortable with git. It comes pre-installed on most linux releases and is already installed on the CSE machines. To check if git is installed on your local machine use the command

```
git status
```

If it is installed you will see something like

fatal: Not a git repository (or any of the parent directories):

If you do not have git installed, you will see something like

bash: git: command not recognized

If this is the case, you will have to set it up using the following instructions

- **Linux** - Follow instructions at <https://git-scm.com/download/linux>
- **Mac** - brew install git

Once it has been installed, we also have to configure git.

Instructions:

1. Try `git status` to check whether git is installed
 - If not, follow one of the above instructions depending on your OS
2. Configure git if you have not used it before with the following commands (including the quotes)

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

```
git config --global user.email "github_email@example.com"
```

1. Create your project repo

Creating and Cloning a **repo** (a repo is just a directory that is linked with git) is how the codebase is linked from GitHub to your local computer so changes you make can be saved and shared with others. It is the final step before you can start making changes and contributing. A repo can be cloned at any time by someone who has access, so they can start working whenever they want. When a repo is cloned, all code that is uploaded on the server is copied to a desired location on your local machine.

This is the process you will follow get create your team and project repo. Make sure only one member of the team is doing this to avoid multiple repos per team.

Instructions:

1. Go to <https://cgi.cse.unsw.edu.au/~se3011/github/run.cgi>
NOTE: You won't be able to access the page if you have not accepted the invitation to join the organisation.
2. Select Teams and under Tutorial select **W09A**. This is the parent project repo.
3. Add your team name and click **Create**
4. Follow the link that is shown in the green banner flashed at the top of the screen to go to the place where you can manage your team
5. Use Members tab to add other members of your team.

NOTE: You won't be able to invite students if they are not registered to join the organisation.

6. Go to Repository home page (<https://github.com/unsw-se3011>) and create new repository for your project. **Make sure it is set to Private.**
7. Go back to team page (<https://github.com/orgs/unsw-se3011/teams/<team-name>/repositories>) and add the created repository.
8. Click on the **Clone or download** button.



9. If the title for the dropdown box is *Clone with HTTPS* click on the **Use SSH** link on the right. The box should look like the below

Clone with SSH ? Use HTTPS

Use an SSH key and passphrase from account.

`git@github.com:cs1531-Admin/lab01-Githi`

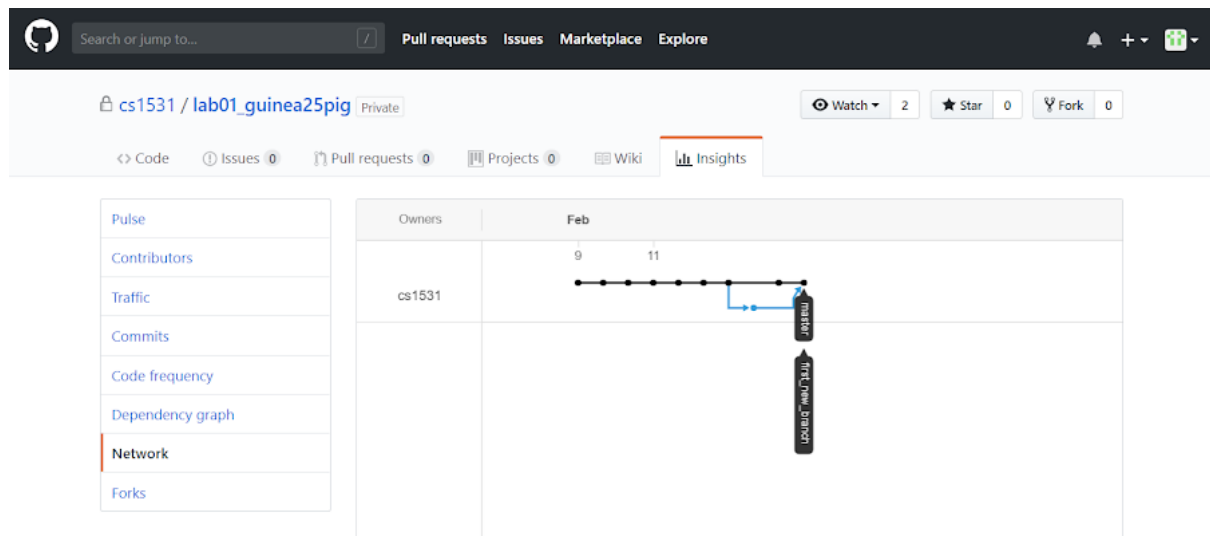
Download ZIP

10. Copy the link in the text box
11. Open a Terminal and navigate to the folder you want the lab to be in
12. `git clone [link]` (Replace [link] with the copied link from above step)
13. Type `ls` to ensure the folder has been copied correctly
14. `cd [repo_name]` to navigate into the cloned repo
15. Type `ls` again to see the copied files.

Quick Tip: Viewing your Git history on GitHub

GitHub provides a decent visualisation of your repo's commit history, which you can access at `Your Repo > Insights > Network` as shown below. Your initial history

will look different to what is displayed here, but watch it evolve as you walkthrough each of the steps below (refresh the page after every pushcommand).



2. Make your first commit

Now that you have cloned the repo, you are ready to work on the codebase locally.

A commit is an update of the remote (on the GitHub's server) state of the repository. It saves changes that you have made and can be given a message to describe those changes. A good use of git involves a lot of commits with detailed messages.

Before you can commit, you have to do what is called staging your changes, which effectively tells git what changes you actually want to commit and what changes you don't at the moment.

Commits are often followed by pushing which is how git actually uploads your commits to the remote server.

The command to commit and push are as follows:

```
git add [files_to_commit] # Stage
```

```
git commit -m "Detailed message describing the changes" # Commit
```

```
git push # Push
```

Summary:

1. Add a new file called `first.txt` in the repo directory
2. Add a line of text to the `first.txt` file using your favourite text editor and save.
3. Go back to your terminal and enter the following commands:

```
git add first.txt`
```

```
git commit -m "Added a line to the first file"
```

```
git push
```

4. MAKE SURE YOU UNDERSTAND THE PURPOSE OF EACH OF THE 3 ABOVE COMMANDS! If you are unsure about any of them, ask your tutor or rewatch the tutorial videos.
5. Go back to GitHub and confirm that your changes have been pushed to the server.

3. Do your first pull

Usually when you are using git, it is in a team. That means that you will not be the only one who is making the changes. If someone else makes a change and pushes it to the server, your local repo will not have the most up to date version of the files. Luckily, git makes it easy to update our local copy with the `git pull` command.

This command checks the remote server that your local repo is linked to and makes sure that all of your files are up to date. This ensures that you don't accidentally do things like implement the same thing someone else has already done and also lets you use other peoples work (eg new functions) when developing.

Pulling regularly is one of the **most important** practices in git!

Unfortunately, at the moment you are just working individually. But GitHub still gives us a nice way to practice a `git pull`.

Summary:

1. View your repo on GitHub.
2. Click on the `first.txt` file
3. Click the small 'edit' pencil icon in the top right
4. Make any change you want to this file and click the **Commit Changes** button at the bottom of the screen
5. This will have changed the `first.txt` file on the server but not on your local environment (check it!). To fetch these changes use the `git pull` command from your terminal
6. Confirm that your version of `first.txt` now has the changes you made on the web page

4. Create your first branch

Branches are a vital part of git and are used so people can work on separate parts of the codebase and not interfere with one another or risk breaking a product that is

visible to the client. Breaking something on one branch does not have an impact on any other.

Good use of git will involve separating parts of the project that can be worked on separately and having them in their own feature branch. These branches can then be merged when they are ready.

Useful commands for branches:

```
git checkout -b [new_branch_name] # Create a new branch and switch to it
git branch                        # List all current branches
git checkout [branch_name]       # Switch to an existing branch
```

Summary:

1. Make your new branch with: `git checkout -b first_new_branch`
2. List your branches to see that you have indeed swapped (use the above commands)
3. Make another change to the `first.txt` file
4. Try to push your changes to the server using the commands you learnt in the *Make your first commit* section
5. The above step should have given you the following error:
– fatal: The current branch `first_new_branch` has no upstream branch.
This means that the branch you tried to make a change on doesn't exist on the server yet which makes sense because we only created it on our local machine.
6. To fix this, we need to add a copy of our branch on the server and link them up so git knows that this new branch maps to a corresponding branch on the server
7. `git push -u origin first_new_branch`

NOTE: The final step above must be run on the 1st push to every new branch that you have created on your local machine. After you have run this once, you should go back to simply using `git push`

5. Merge your two branches

Merging branches is used to combine the work done on two different branches and is where git's magic really comes in. Git will compare the changes done on both branches and decide (based on what changes were done to what sections of the file and when) what to keep. Merges are most often done when a feature branch is complete and ready to be integrated with the master branch.

Since we have finished all that we are going to do (and think there are no bugs) on our *first_new_branch* we can merge it back into master. It is a strong recommendation to have a version of the code that at least runs on master so people are not completely blocked. (DO NOT PUSH BROKEN CODE TO MASTER)

Another recommendation is to merge master into your branch before merging your branch into master as this will ensure that any merge into master will go smoothly.

Commands for merging two branches

```
git merge [target] # Merge the target branch into current
```

NOTE: A successful merge automatically uses the commits from the source branch. This means that the commits have already been made, you just need to push these to the server (`git push`)

Summary:

1. Switch back to the master branch using one of the commands from the above section
2. Merge in the changes you made in the other branch
`git merge first_new_branch`
3. Push the successful merge to the server to update the master branch on the server

6. Engineer a merge conflict :(

Merge conflicts are the one necessary downside to git. Luckily, they can be avoided most of the time through good use of techniques like branches and regular commits, pushes and pulls. They happen when git cannot work out which particular change to a file you really want.

For this step we will engineer one so you can get a taste of what they are, how they occur and how to fix them. This will be the LAST time you will want one. The process may seem involved but it is quite common when multiple people are working at a time.

Summary: (All commands have been presented above)

1. Add a line to the top of the `first.txt` file (on *master branch*)
2. Add, commit and push your changes
3. Switch to your *first_new_branch*
4. Add a different line to the top of the `first.txt` file
5. Add, commit and push your changes
6. Merge master into your current branch
7. This sequence of steps should made a merge conflict at the top of the `first.txt` with the following output
Auto-merging `first.txt`
CONFLICT (content): Merge conflict in `first.txt`
Automatic merge failed; fix conflicts and then commit the result.

7. Resolve your merge conflict :)

Resolving a merge conflict is as simple as editing the file normally, choosing what you want to have in the places git wasn't sure.

This is a very simple example, but merge conflicts can be large and in many different places across a file/repo. If possible, avoid merge conflicts. This can be done by regularly pulling from the server to update your local copy and by making your branches in such a way that they handle only one feature/section of the code.

A merge conflict is physically shown in the file in which it occurs.
<<<<<<< marks the beginning of the conflicting changes made on the **current** (merged into) branch.
===== marks the beginning of the conflicting changes made on the **target** (merged) branch.
>>>>>> marks the end of the conflict zone.

E.g.,

This line could be merged automatically.

There was no change here either

```
<<<<<<< current:sample.txt
```

Merges are too hard. This change was on the 'merged into' branch

```
=====
```

Merges are easy. This change was made on the 'merged' branch

```
>>>>>> target:sample.txt
```

This is another line that could be merged automatically

This above example could be solved in many ways, one way would be to just use the changes made on the target branch and delete those made on the current branch. Once we have decided on this we just need to remove the syntax. The resolved file would be as follows

This line could be merged automatically.

There was no change here either

Merges are easy. This change was made on the 'merged' branch

This is another line that could be merged automatically

We would then just commit the resolved file and the merge conflict is finished!

Summary:

1. Open the `first.txt` file and decide which (or both) changes you want to keep
2. Remove the merge conflict syntax
3. Add, commit and push the resolved merge conflict
4. Set up `first.txt` on *master* branch - you see that merging *master* branch into *first_new_branch* has only affected *first_new_branch*.
5. Now, merge *first_new_branch* into *master* to synchronise the two branches (this step should not cause any conflicts), then push the changes.
6. Check your git history visualisation on GitHub :)

Testing

You can run the `test_git.sh` file to check whether you have done most of the git exercises. Make sure you checkout the master branch before running this script.