Having your project stored remotely has many advantages.
1) You are not restrained to work on a particular computer where you have that local repository stored;
2) You can always retrieve your project so long you're connected to the Internet.
3) People in your team can keep up to date, and submit changes to a remote location.
4) You don't lose things after a hard disk failure.
5) and many others.

1. Preparation

Have a remote Git repository that you want to clone from already? Cool, skip this step!

Find any online, free Git repository hosting, for example:
https://github.com/

Make an account, and create a new repository. For **github**, you have the choice to initialise it with a **README** (and a **.gitignore** file tailored to your particular application, more on this later), allowing you to **git clone** from it straight after. I don't have any existing remote Git repositories, so I just made a new one for my PhD thesis, named **thesis**.

2. Clone the remote repository

Navigate to the parent directory where you want this new Git repository stored. Open the Git Bash and do **git clone [url] [directory name]**.

```
$ git clone git://github.com/xxxxxx/thesis.git
Initialized empty Git repository in c:/Users/rrd09/Documents/thesis/.git/
Cloning into 'thesis' ...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (4/4), done.
```

3. Add some files

If you already have some files that you want to track, you can start adding them using **git add [file name]**, assuming that they are now in your local working directory. Here I add some of my Latex files.

```
rrd09@XXXXX ~/Documents/thesis (master)
$ git add dissertation.tex
```

You may see this warning message: **warning: LF will be replaced by CRLF in dissertation.tex. The file will have its original line endings in your working directory**, which just tells you that Git will unify the "end of line" representations in Unix (LF) and Windows (CRLF), nothing to worry about.

Git supports wildcards too:

```
$ git add *.tex
```

and you can also add folders, denoted by the tailing forward slash **/**:

```
$ git add introduction/
$ git add background*/
```

3. Once you finished adding files, commit the change:

```
$ git commit -m 'Initial version of thesis'
[master (root-commit) 492a8b4] Initial version of thsis
 48 files changed, 11944 insertions(+)
 create mode 100644 dissertation.tex
 ...
 create mode background_fs/001.aux
 create mode background_fs/001.tex
 create mode background_nim/001.aux
 create mode background_nim/001.tex
 ...
 create mode 100644 introduction/001.aux
 create mode 100644 introduction/001.texx
 ...
```

Note that you must provide a (hopefully meaningful) commit message using **-m '[message]'**, which will help you to keep track of these changes in the future.

4. Check the status of your repository

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       abstract.aux
#       dissertation.aux
#       ...
#       dissertation.synctex.gz
#       ...
```

Here I am intentionally not adding some files as they are intermediate outputs of the Latex editor. Similarly, you may not want to add your complier outputs, temporary files, etc.

5. Un-track some files

There are times when you accidentally tracked some files, like **background_fs/001.aux**, you may un-track them by doing **git rm --cached [file name]**.

```
$ git rm --cached back*/*.aux
rm 'background_fs/001.aux'
rm 'background_nim/001.aux'
```

and lets commit these changes:

```
$ git commit –m 'Removed tracking on some unnecessary files'
[master fc9ce44] Removed tracking on some unnecessary files'
 12 files changed, 861 deletions(-)
 ...
 delete mode 100644 background_fs/001.aux
 delete mode 100644 background_nim/001.aux
 ...
```

6. Ignoring files

To stop Git from worrying about these files all together, we can add them to the ignore list. For this, you need to make a new file named **.gitignore**, and populate it with some patterns, or file names. For example:

```
$ echo "*.aux" > .gitignore
$ echo "*.gz" >> .gitignore
```

will add any file matching the **.aux** extension to the ignore list, regardless of where they are in the entire repository. Regular expressions such as ***.[oa]**, matching any file ending in **.o** or **.a**, and ***~**, matching anything ending in **~**, are all accepted.

After adding some files in **.gitignore**, we have:

```
$ git status
# On branch master
# Your branch is ahead of ‘origin/master’ by 2 commits.
#   (use “git push” to publish your local commits)
#
# Untracked files:
#   (use “git add <file>...” to include in what will be committed)
#
#       .gitignore
nothing added to commit but untracked files present (use “git add” to track)
```

The ignore list can/should of course, also be added to the repository.

7. Push to the remote server

Do this only if you have the permission to publish the changes, of course. The command is **`git push [url]`**. Below is the final output after I pushed my local folder up the first time.

```
$ git push https://gitbub.com/xxxxxx/thesis.git
warning: ...
...
Username for ‘https://github.com’:
Password for ‘https://github.com’:
Counting objects: 77, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (64/64), done.
Writing objects: 100% (76/76), 4.41MiB | 749.00 KiB/s, done.
Total 76 (delta 4), reused 0 (delta 0)
To https://github.com/xxxxxx/thesis.git
   d173032..4f7da05  master -> master
```

Note that you need the **`https`** protocol here since authentication is required.