



Portfolio 5

Modul Name: Software Engineering

Year 2

Student Name: Mohamed Wisam

Student ID: A2569

Table of Contents

VERSION CONTROL	3
WHAT IS MERGING?	6
BRANCHING	7
CHANGESET	7
REVERTING	7
GITHUB	8
HOW TO USE GITHUB	8

VERSION CONTROL

A version control system is a repository of files, saved and managed with monitored access which includes mostly source code of computer programs. Version Control software is used to track every change made to the code along with who made it, when and why. It is used for developing in a team.

Version control is essential for any form of team, distributed and collabrated development. It holds the information about the previous versions, so it uses it to compare with old versions as well as the changes made back. It allows multiple users and branches, keeping the changes made with the information about the person who made it. In case of any mistake, the system can revert back, recover the files. Multiple people can work on a same file with the help of version control.

In the software engineering community, version control has been widely studied and developed. It is a highly sophisticated system which is well supported by different operating systems and also very stable. There are several version control services such as GitHub, SourceForge and GoogleCode. These services have a wide range of functionality from archiving, release downlaods to bug trackers. Version control software also allows the user to compare the changes made overtime, including who created the issues and when.

There are three types of version control systems:

Local Version Control System

More simplicity, but very error prone.

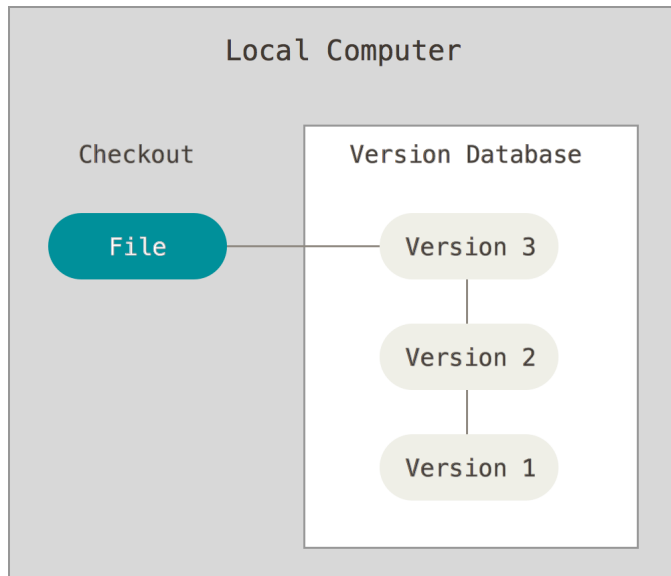


Figure 1: Local Version Control System

Centralized Version Control System

This system uses a centralized server to hold up all the files, and the clients.

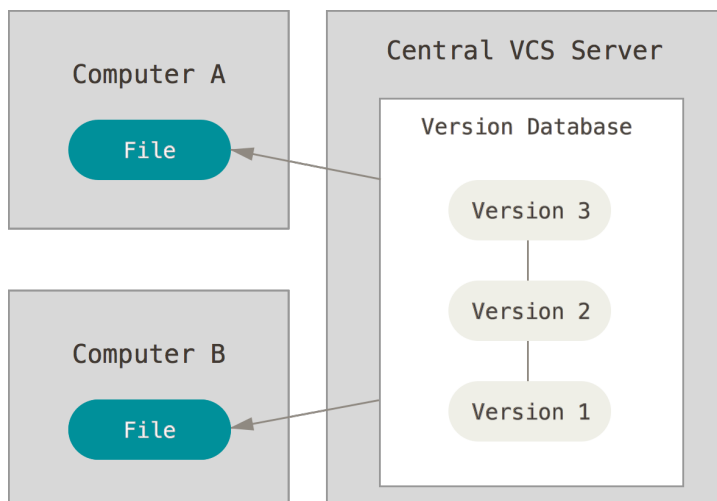


Figure 2: Centralized Version Control System

Distributed Version Control System

In DVCS, every client have copies of the full repository. This gives further protection for the files as even if the server dies, all the files from the clients are copied back.

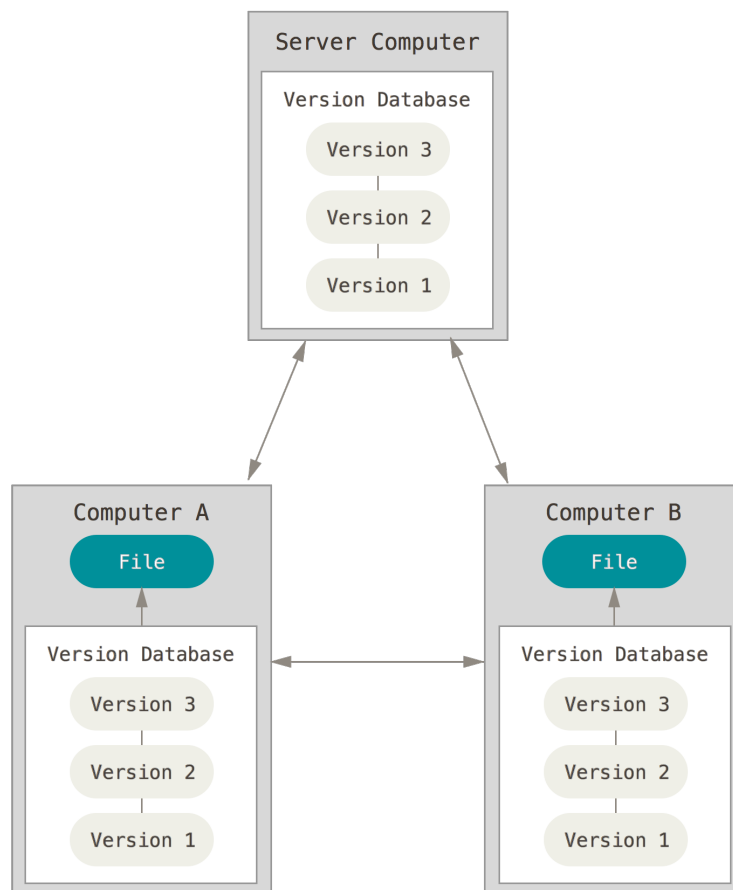


Figure 3: DVCS

WHAT IS MERGING?

Merging is a combination of two different files that is being edited and updated from two different users, on different systems. Merging allows it to match with each other, thus allowing to work together without conflict. Merging will usually retrieve a code that was edited by one user and process is to combine into a different file and on to a code. The process of combining these functions to make a single entity is called **merging**.

Mergin is used when working on groups. Once the programmer is done with the code he/she would need to add it to the master file. Every change to the file will be recorded, having a history of all the changes made. Each change will be done to the branch and it can be merged into the master file. It is sort of an updating the master file with changes. In Some cases, version control might have problems figuring out the changes to be done to the revisions. Merging can be performed automatically, considering there is sufficient information about the changes made, history. It also depends on whether the changes does not conflict. Almost all version control softwares include merge capability.

There are two types of merge. Automatic and manual merge. Automatic merging happens when the version control software reconciles that changes have been made simultaneously. Manual mergin occurs when two or more people have to do the merging using version control software but different files. This is needed in most cases when the two systems have different versions. In this case, it will be needed to merge the configuration files and the changes done. This is two way merging.

BRANCHING

In version control, branching is when a duplication of the file is made so that modification can be done altogether. Branching is used so that the main file can not be edited, mostly in experimental purpose. In more detail, using branches it creates copies of the files. The user commits to those files while others commit the changes to the main file. In this way, the changes does not affect the users.

CHANGESET

Changesets is a single container that contain source file and folder revision which include the add, rename, remove, edit and moves. A combination of the source file will produce a single code from the server that usually allow the user to merge other file by using the code which generate from the changesets. A revision number will be given to each changeset so that the user can revisit the changes they did. It also includes more information about who committed, when and the changes made including the directories.

REVERTING

Reverting the changes made is another ability of most version control softwares. It uses the revision files to go back. Reverting is extremely important in case of changes to be made, track back and protect from unnecessary changes already made.

GitHub

GitHub is a web based repository hosting service that offers revision control and source code management service, as well as other features. GitHub is highly popular among developers as even famous company's such as Google and Twitter uses it. GitHub is the best place for a developer to share the source code. It has got all the features of version control along with other features.

HOW TO USE GitHub


To use github, first of all the user must register and create a new repository.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name


 wisam87 ▾

 /


GitTesting ✓

Great repository names are short and memorable. Need inspiration? How about [laughing-octo-guide](#).

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.


☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

 |

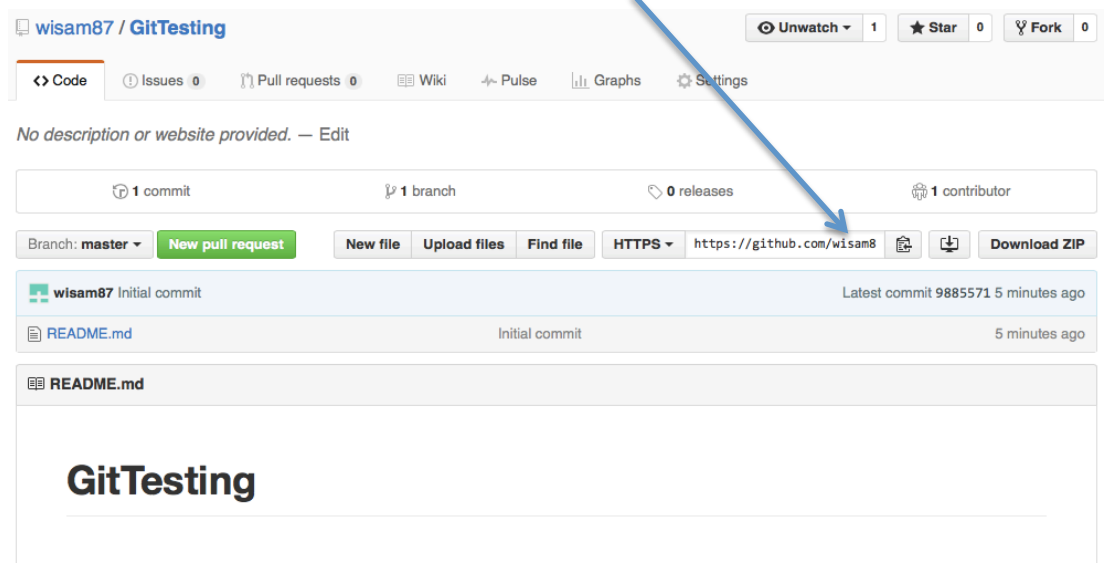
Add a license: **None** ▾



Create repository

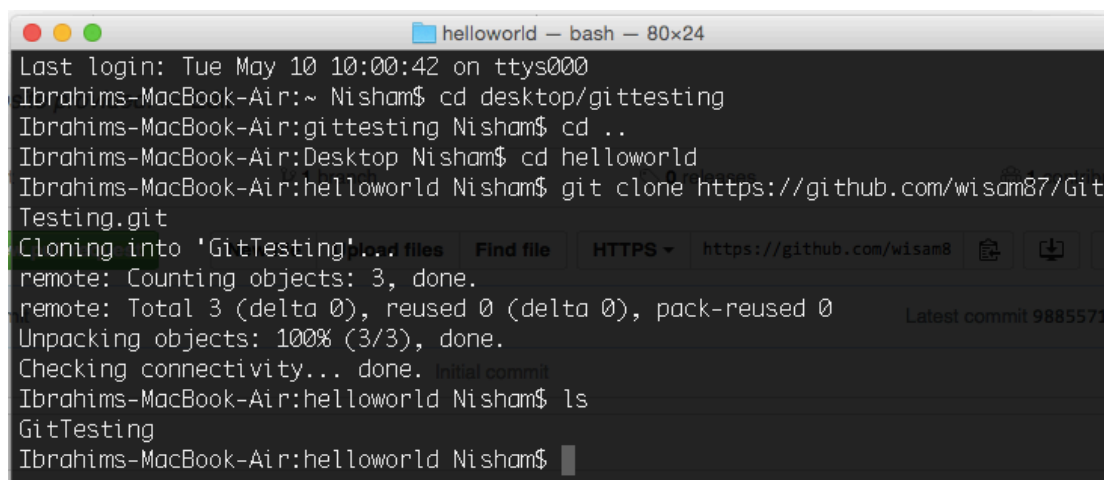
Figure 4: Creating a Repository

After creating the repository, copy the link.



For Mac users, open up the terminal, select a folder and enter the command `git clone <link to the repository>`

`git clone https://github.com/wisam87/GitTesting.git`



It creates a folder named as the repository, inside the folder selected.
Just using the link, other users can also use the link to share the folder.
as show below,

```
secondfolder — bash — 85x27
Last login: Tue May 10 10:07:02 on ttys001
Ibrahims-MacBook-Air:~ Nisham$ cd desktop/secondfolder
Ibrahims-MacBook-Air:secondfolder Nisham$ git clone https://github.com/wisam87/GitTesting.git
Cloning into 'GitTesting'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
Ibrahims-MacBook-Air:secondfolder Nisham$
```

Now both user's created there cpp files in the folder,

```
main.cpp
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello everybody. Wisam File" << endl;
7 }
```

Line 6, Column 41 Tab Size: 4 C++

```
main2.cpp
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "This is Ali's File" << endl;
7 }
```

Line 6, Column 42 Tab Size: 4 C++

Now use command *git status* to see the difference from the repository and the file in the computer.

```
fatal: destination path 'GitTesting' already exists and is not an empty director
y.
Ibrahims-MacBook-Air:helloworld Nisham$ git clone https://github.com/wisam87/Git
Testing.git
Cloning into 'GitTesting'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
Ibrahims-MacBook-Air:helloworld Nisham$ git status
fatal: Not a git repository (or any of the parent directories): .git
Ibrahims-MacBook-Air:helloworld Nisham$ ls
GitTesting
Ibrahims-MacBook-Air:helloworld Nisham$ cd GitTesting
Ibrahims-MacBook-Air:GitTesting Nisham$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    main.cpp

nothing added to commit but untracked files present (use "git add" to track)
Ibrahims-MacBook-Air:GitTesting Nisham$
```

Now the user must add the new file shown as untracked files above.

To do this use the command `git add <filename>`

git add main.cpp

```
Ibrahims-MacBook-Air:GitTesting Nisham$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file:   main.cpp

Ibrahims-MacBook-Air:GitTesting Nisham$
```

now it shows the file added and changes to be committed.

Now the user must commit the file, lock it to the repository.

git commit -m "Message"

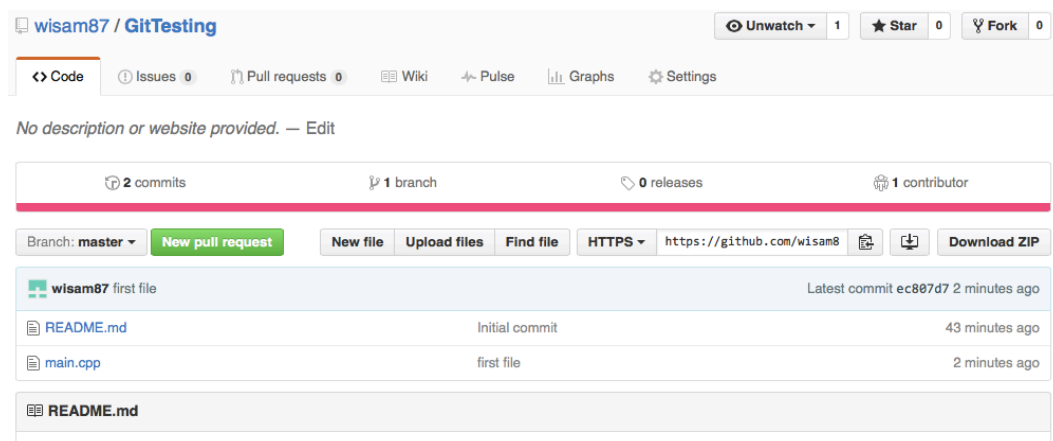
```
Ibrahims-MacBook-Air:GitTesting Nisham$ git commit -m "first file"
[master ec807d7] first file
1 file changed, 7 insertions(+)
 create mode 100644 main.cpp
Ibrahims-MacBook-Air:GitTesting Nisham$
```

Still the changes the user have made is not saved to github.com.

Now use the command *git push* to do that.

```
Username for 'https://github.com': wisam87
Password for 'https://wisam87@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 366 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/wisam87/GitTesting.git
9885571..ec807d7 master -> master
Ibrahims-MacBook-Air:GitTesting Nisham$
```

Now the file is saved and can be seen on github.com



Now when the second user uses the command *git pull* the new file *main.cpp* is downloaded into there folder

```
Ibrahims-MacBook-Air:gitTesting Nisham$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    main2.cpp

nothing added to commit but untracked files present (use "git add" to track)
Ibrahims-MacBook-Air:gitTesting Nisham$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/wisam87/GitTesting
   9885571..ec807d7  master    -> origin/master
Updating 9885571..ec807d7
Fast-forward
   main.cpp | 7 ++++++
   1 file changed, 7 insertions(+)
   create mode 100644 main.cpp
Ibrahims-MacBook-Air:gitTesting Nisham$ ls
README.md      main.cpp      main2.cpp
Ibrahims-MacBook-Air:gitTesting Nisham$
```

Using *git status* it shows the modified files. Now add all the files using command: *git add <filename>* or *git add -A* (this is to add all files)

```
Your branch is up-to-date with 'origin/master'.
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:   main.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    main2.cpp

no changes added to commit (use "git add" and/or "git commit -a")
Ibrahims-MacBook-Air:gitTesting Nisham$ git add -A
Ibrahims-MacBook-Air:gitTesting Nisham$
```

Now the second user commits and push the files with the changes made.

```
GitTesting — bash — 80x24

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 480 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://github.com/wisam87/GitTesting.git
    ec807d7..bb33047  master -> master
Ibrahims-MacBook-Air:gitTesting Nisham$
```

Now the first user uses the command *git pull*
and the changes are saved to his file automatically as well as adding the
main2.cpp created and committed by the second user

```
main.cpp

#include <iostream>

using namespace std;

int main() {
    cout << "Hello everybody. Wisam File" << endl;
    cout << "Changes made by second user" << endl;
}
```

In conclusion it is how GitHub works.