A red and white logo

Description automatically generated with low confidence A picture containing text, clipart

Description automatically generated

**Git Practices for Daily Use**

In this article we will learn basic principles about version control, and we will carry out a simple git practise together.

What is version control?

Version control in software engineering is known as source control. It is the methods used by software development and IT operations teams to track and manage changes to software and documentation. Multi player collaboration requires coordination and forces teams to record what modifications done when and by who so that possible confusions can be avoided, in case of failures old versions of the code can be re cycled and DevOps teams can achieve development time reduction and increase deployment success rate. Versioning also allows many teams to work on same software package's several different features concurrently.

What is Git?

Git is a free and open-source version control system designed to handle every software development projects from smallest to largest with speed, reliability and efficiency. Original developed by Linus Torvalds, widely used individuals and organisations since its release in 2005. Please go to <https://git-scm.com/about> for more information about git.

To start using Git please go to https://git-scm.com/downloads to download. Please choose your computer's operating system and follow the step-by-step simple instructions on the page. Once you successfully downloaded Git on your computer you good to go.

Before start using Git please ensure to configure your info as follows:

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

$ git config --global user.email "<Your Email>@gmail.com"

You can double check accuracy of your name and email entries with:

$ git config --global user.name

$ git config --global user.email

Let's start the practise by creating a folder on our desktop:

$ cd Desktop/

$ mkdir git-project

$ cd git-project

Let's initiate git project:

$ git init

In order to display all files in the folder:

$ ls -a

Now it is time to create our first file:

$ vim first.py

Switch to insert more by pressing "i" key and type

print("Hello World")

Then switch back by pressing "esc" key and once you save your work and quit the file by pressing ":wq" you can check the file:

$ ls

You can check content of the file:

$ cat first.py

You are now ready to place the file in your staging area:

$ git add .

Let's place the file in your local repository:

$ git commit -m "First commit"

Output should look like:

[master (root-commit) 6753d94] First commit

1 file changed, 1 insertion(+)

create mode 100644 first.py

List the existing versions:

$ git log

Output should look similar to:

commit 6753d9404f13bfc677e0354e269d6b129ae8abed (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:45:22 2021 +0000

First commit

To see the changes:

$ git status

output:

On branch master

nothing to commit, working tree clean

Let's add another file to our repository:

vim second.txt

This is the second file of git project.

To see the changes:

$ git status

output:

On branch master

Untracked files:

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

second.txt

nothing added to commit but untracked files present (use “git add” to track)

To add newly created file to staging area:

$ git add second.txt

$ git status

output:

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: second.txt

To commit new file to repository:

$ git commit -m "second.txt added"

output:

[master 113300f] second.txt added

Author: M. A. <m.a@macbookpro.finspiretech>

1 file changed, 1 insertion(+)

create mode 100644 second.txt

$ git status

output:

On branch master

nothing to commit, working tree clean

Now let's list all the version:

$ git log

output:

commit 113300f0873ecf9aabfa075ed3ad0f841004fe6d (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:56:06 2021 +0000

second.txt added

commit 6753d9404f13bfc677e0354e269d6b129ae8abed

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:45:22 2021 +0000

First commit

Let's make some changes to the first file to experience how the version control will be displayed:

$ vim first.py

print("This is the modified Hello World")

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: first.py

no changes added to commit (use "git add" and/or "git commit -a")

In order to send new version to statging area and to repositoy and display the history:

$ git add .

$ git commit -m "Modified version of first.py"

$ git log

output:

commit 199096ef721ed01a8a59f1d8551d79af02def86b (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:05:10 2021 +0000

Modified version of first.py

commit 113300f0873ecf9aabfa075ed3ad0f841004fe6d

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:56:06 2021 +0000

second.txt added

commit 6753d9404f13bfc677e0354e269d6b129ae8abed

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:45:22 2021 +0000

First commit

$ git status

output:

On branch master

nothing to commit, working tree clean

Let's make some changes to the other files:

$ vim second.txt

This is the modification to second file of git project

$ vim first.py

print("This is the second modification to Hello World")

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: first.py

modified: second.txt

no changes added to commit (use "git add" and/or "git commit -a")

Let's send the new versions to statging area:

$ git add .

Let's see the difference between working file versions in staging area:

$ git diff

output:

diff --git a/first.py b/first.py

index edb61be..402934e 100644

--- a/first.py

+++ b/first.py

@@ -1 +1 @@

-print("This is the modified Hello World")

+print("This is the second modification to Hello World")

diff --git a/second.txt b/second.txt

index d93d3fd..660a419 100644

--- a/second.txt

+++ b/second.txt

@@ -1 +1 @@

-this is the second file of git project

+This is the modification to second file of git project

Let's commit the changes:

$ git commit -m "changes to both files"

output:

[master 41e8ee5] changes to both files

Committer: M. A. <m.a@macbookpro.finspiretech>

2 files changed, 2 insertions(+), 2 deletions(-)

Let's delete one fo the files and check status

$ rm second.txt

$ ls

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: second.txt

no changes added to commit (use "git add" and/or "git commit -a")

We can send the change to staging area (either by git add command or) by git rm command as follows:

$ git rm second.txt

output:

rm 'second.txt'

$ git status

output:

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: second.txt

$ git commit -m "second.txt deleted"

output:

[master caf7fe7] second.txt deleted

Committer: M. A. <m.a@macbookpro.finspiretech>

1 file changed, 1 deletion(-)

delete mode 100644 second.txt

So far we have been working with files. Let's start working with folders:

$ mkdir folder1

$ cd folder1

Create multiple files:

$ touch third.txt fourth.java fifth.js

$ cd ..

Let's send the change to local git repositor:

$ git add .

$ git commit -m "created folder1 and three files in it"

output:

[master 4ebb037] created folder1 and three files in it

Committer: M. A. <m.a@macbookpro.finspiretech>

3 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 folder1/fifth.js

create mode 100644 folder1/fourth.java

create mode 100644 folder1/third.txt

Let's now delete the folder:

$ git rm -r folder1

output:

rm 'folder1/fifth.js'

rm 'folder1/fourth.java'

rm 'folder1/third.txt'

$ git status

output:

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: folder1/fifth.js

deleted: folder1/fourth.java

deleted: folder1/third.txt

$ git commit -m "folder1 deleted"

output:

[master cfd48be] folder1 deleted

Committer: M. A. <m.a@macbookpro.finspiretech>

3 files changed, 0 insertions(+), 0 deletions(-)

delete mode 100644 folder1/fifth.js

delete mode 100644 folder1/fourth.java

delete mode 100644 folder1/third.txt

(base) macbookpro:git-project m.a$ git commit -m "folder1 deleted"

On branch master

nothing to commit, working tree clean

Now we will try to undo modification records. Let's use first.py file:

$ vim first.py

Let's delete entire content of the file as if we were making an unintentional modification. Check to make sure content of the file deleted and check git status:

$ cat first.py

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: first.py

no changes added to commit (use "git add" and/or "git commit -a")

As you can see the message displays actually showing us how to discard changes:

$ git restore first.py

(If above do not work, or prompted otherwise please note you may need to use "git checkout -- first.py" instead on some operating systems)

$ git status

output:

On branch master

nothing to commit, working tree clean

$ cat first.py

Output shows that modifications restored:

Print ("This is the second modification to Hello World")

Let's make more files and delete the very first file:

$ vim sixth.txt

This is sixth file for git experiment.

$ vim seventh.txt

This is seventh file for git experiment.

$ rm first.py

$ ls

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: first.py

Untracked files:

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

seventh.txt

sixth.txt

no changes added to commit (use "git add" and/or "git commit -a")

As you can see that first.py file has been deleted. Let's salvage the deleted file back:

$ git restore first.py

$ ls

$ git status

outcome:

On branch master

Untracked files:

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

seventh.txt

sixth.txt

nothing added to commit but untracked files present (use "git add" to track)

Now we will restore a file which is already sent to staging area. Let's replace content of first.py file with "..."

$ vim sixth.txt

$ vim first.py

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: first.py

Untracked files:

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

seventh.txt

sixth.txt

no changes added to commit (use "git add" and/or "git commit -a")

Let's send the changed file to staging area:

$ git add first.py

$ git status

$ git restore --staged first.py

(Some operating systems may require you to use "git reset HEAD first.py" command instead)

$ git status

output:

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: first.py

Untracked files:

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

seventh.txt

sixth.txt

no changes added to commit (use "git add" and/or "git commit -a")

Now we have restored delete command but we have not yet restored the file:

$ git restore first.py

We can roll back to previous version that has been recorded by earlier commits. Let's now practise changing version:

$ git log

output:

commit cfd48be67bc416defe694c866fa6d67fa68f94f3 (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:26:14 2021 +0000

folder1 deleted

commit 4ebb037dd1525c8f1ee1ce8444d20e0aba54050b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:22:54 2021 +0000

created folder1 and three files in it

commit caf7fe7d8224476c796454ad0c274860ee62334e

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:25:54 2021 +0000

second.txt deleted

commit 41e8ee52f6d42d407307709a292610ade1b4940f

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:20:37 2021 +0000

changes to both files

commit 199096ef721ed01a8a59f1d8551d79af02def86b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:05:10 2021 +0000

Modified version of first.py

commit 113300f0873ecf9aabfa075ed3ad0f841004fe6d

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:56:06 2021 +0000

second.txt added

:...skipping...

commit cfd48be67bc416defe694c866fa6d67fa68f94f3 (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:26:14 2021 +0000

folder1 deleted

commit 4ebb037dd1525c8f1ee1ce8444d20e0aba54050b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:22:54 2021 +0000

created folder1 and three files in it

commit caf7fe7d8224476c796454ad0c274860ee62334e

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:25:54 2021 +0000

second.txt deleted

commit 41e8ee52f6d42d407307709a292610ade1b4940f

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:20:37 2021 +0000

changes to both files

commit 199096ef721ed01a8a59f1d8551d79af02def86b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:05:10 2021 +0000

Modified version of first.py

commit 113300f0873ecf9aabfa075ed3ad0f841004fe6d

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:56:06 2021 +0000

second.txt added

commit 6753d9404f13bfc677e0354e269d6b129ae8abed

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:45:22 2021 +0000

:...skipping...

commit cfd48be67bc416defe694c866fa6d67fa68f94f3 (HEAD -> master)

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:26:14 2021 +0000

folder1 deleted

commit 4ebb037dd1525c8f1ee1ce8444d20e0aba54050b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 17:22:54 2021 +0000

created folder1 and three files in it

commit caf7fe7d8224476c796454ad0c274860ee62334e

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:25:54 2021 +0000

second.txt deleted

commit 41e8ee52f6d42d407307709a292610ade1b4940f

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:20:37 2021 +0000

changes to both files

commit 199096ef721ed01a8a59f1d8551d79af02def86b

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 15:05:10 2021 +0000

Modified version of first.py

commit 113300f0873ecf9aabfa075ed3ad0f841004fe6d

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:56:06 2021 +0000

second.txt added

commit 6753d9404f13bfc677e0354e269d6b129ae8abed

Author: M. A. <m.a@macbookpro.finspiretech>

Date: Sun Feb 21 14:45:22 2021 +0000

First commit

All versions listed above. Now we can restore any version we need using hash code (git checkout <hash\_code> -- .). You can either use entire code or first 7 digit will also do.

$ git checkout 41e8ee52f6d42d407307709a292610ade1b4940f -- .

(If you want to roll back only one specific file then you should use file name instead of "." at the end of the command above.)

$ git status

You may need to stage and commit the restored file:

$ git add .

$ git commit -m "Commit after restoring previous version of second.txt"

Creating, switching, merging branches:

$ git branch

output:

\* master

Creating a new branch:

$ git branch new\_branch

$ git branch

output:

\* master

new\_branch

Swithcing to new branch:

$ git checkout new\_branch

output:

Switched to branch 'new\_branch'

$ git branch

output:

master

\* new\_branch

Now we are on the new branch namely new\_branch

$ touch new\_branch\_file.txt

$ git add .

$ git commit -m "new\_branch\_file.txt added"

output:

[new\_branch f3ef229] new\_branch\_file.txt added

Committer: M. A. <m.a@macbookpro.finspiretech>

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 new\_branch\_file.txt

Let's merge the two branches. We have to switch to the master branch first so that we can merge two branches:

$ git checkout master

output:

Switched to branch 'master'

$ git merge new\_branch

output:

Updating 41e573b..f3ef229

Fast-forward

new\_branch\_file.txt | 0

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 new\_branch\_file.txt

Using remote repository:

Go to your Github account, click Repositories, click green new button on top right hand side, and create new repo with name "python" (and do not check README)

Go to Desktop and create a folder named python and a file named text.txt inside:

$ mkdir pythin

$ cd python

$ touch text.txt

$ git init

$ git add .

$ git status

$ git commit -m "text.txt added"

$ git remote add origin https://github.com/<youraccount>/<remote-repo.git>

$ git push -u origin master

Next time your file modified next time you can update your remote repo by:

$ git add .

$ git commit -m "second added"

$ git push

If you already have a Github repository and you would like to use the content of that particular remote repository you can use following command to copy content and connect the local repository as origin to your local repository as follows.

First go to Github repository page click green coloured code button on top right hand side, click HTTPS and copy the content of the box right below HTTPS.

And then go into the directory where you want to copy the Github / remote repository and use:

$ git clone https://github.com/<your github username>/<repository name>.git

output:

Cloning into 'jenkins-maven-project'...

remote: Enumerating objects: 37, done.

remote: Counting objects: 100% (37/37), done.

remote: Compressing objects: 100% (23/23), done.

remote: Total 37 (delta 2), reused 32 (delta 0), pack-reused 0

Receiving objects: 100% (37/37), 7.58 KiB | 3.79 MiB/s, done.

Resolving deltas: 100% (2/2), done.

$ cd jenkins-maven-project

$ ls -al

You will see that among the other files a ‘.git’ file appeared confirming that this folder is a git repository. And from now on modifications at local repository can be uploaded to remote repository by using git push and modifications at remote repository can be downloaded by using git pull commands. Let's practise:

$ vim hello.txt

$ git status

$ git add .

$ git commit -m "added hello.txt"

$ git push

Now you should be able to see new file in your remote repository. Open the file at remote repository and make changes to content of hello.txt. In order to update local repository of the modifications made in remote repository and see content change of hello.txt:

$ git pull

$ cat hello.txt

You are now gone through full cycle of version control basics by creating a folder, staging (indexing), committing to local repository, modifying, rolling back, branching, merging and finally connecting your local repository to remote repository. Also cloning from remote repository and updating modifications to and from remote repository.

There are lot to learn and practise about whole software development life cycle management. Please note growth curve follows the learning curve. Practise make it perfect.

I keep practising.

Author:

M. Altun

21Feb2021, London

DevOps Engineer @ Finspire Technology