Learning Git

A short course on Git and Git hub

CIT Python in Ramadan

Version Control System??

You can think of a version control system (short: “VCS”) as a kind of “database”. It lets you save a snapshot of your complete project at any time you want. When you later take a look at an older snapshot (let‘s start calling it “version”), your VCS shows you exactly how it differed from the previous one.

Git

Helix VCS

Microsoft Team Foundation Server

Subversion

Rational Clear Case

Why Use a Version Control System?

- Collaboration
- Storing Versions (Properly)
- Restoring Previous Versions
- Understanding What Happened
- Backup

Command Line or GUI?

There are two main ways of working with Git:

Command Line Interface (Bash)

GUI application. (Git Hub)

Neither of these are right or wrong.

On the one hand, using a GUI application will make you more efficient and let you access more advanced features that would be too complex on the command line.

On the other hand, however, I recommend learning the basics of Git on the command line, first. It helps you form a deeper understanding of the underlying concepts and makes you independent from any specific GUI application.

Repository

-A kind of database where your VCS stores all the versions and metadata that accumulate in the course of your project.

- In Git, the repository is just a simple hidden folder named ".git" in the root directory of your project.
- Knowing that this folder exists is more than enough.

Local & Remote Repositories

There are two kinds of repositories:

- A “local” repository resides on your local computer, as a “.git” folder inside your project‘s root folder. You are the only person that can work with this repository, by committing changes to it.

- A “remote” repository, in contrast, is typically located on a remote server on the internet or in your local network. No actual working files are associated with a remote repository: it has no working directory but it exclusively consists of the “.git” repository folder. Teams are using remote repositories to share & exchange data: they serve as a common base where everybody can publish their own changes and receive changes from their teammates.

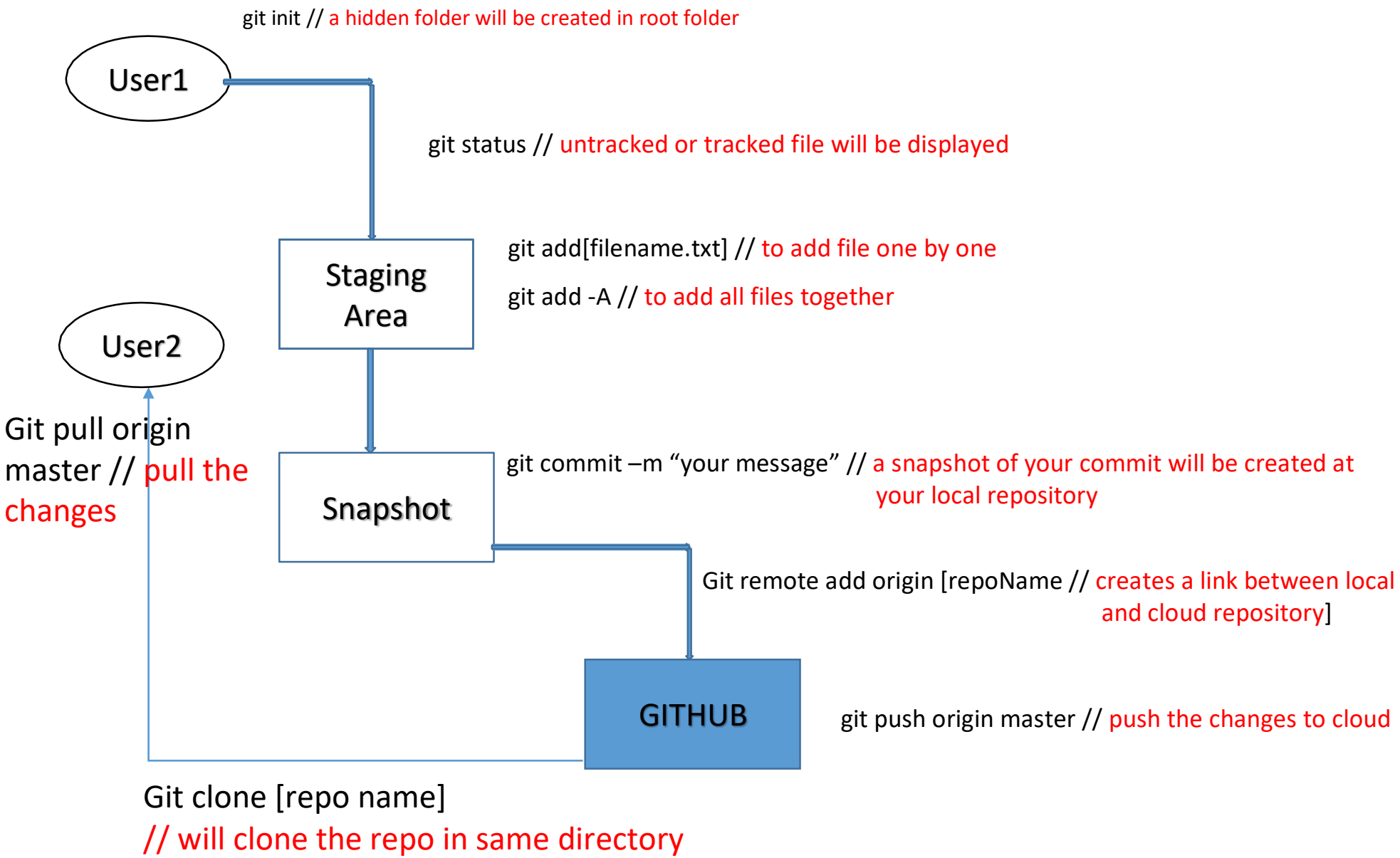
Commit

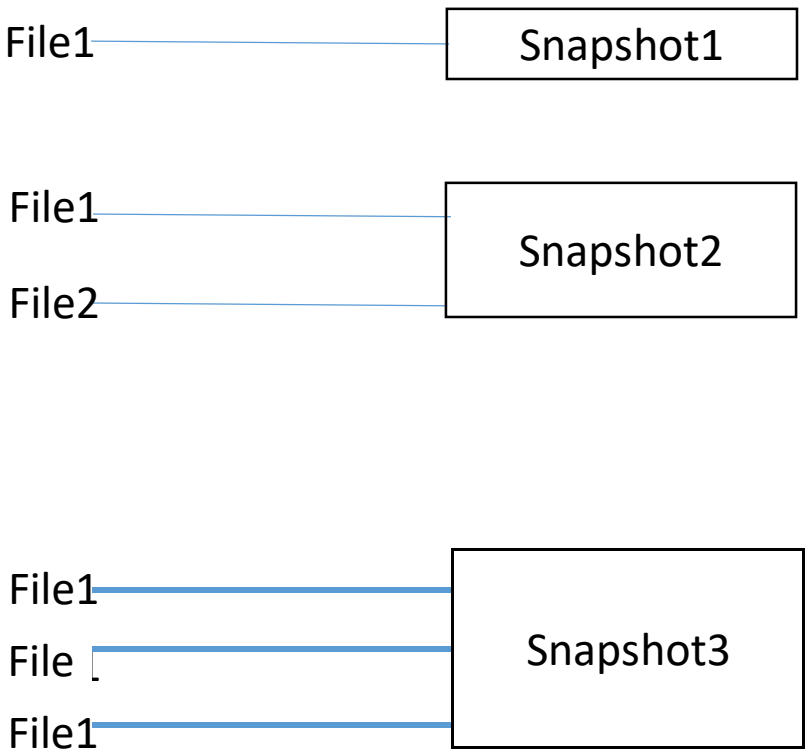
- A commit is a wrapper for a specific set of changes.
- Every set of changes implicitly creates a new, different version of your project.
- Every commit also marks a specific version. It‘s a snapshot of your complete project at that certain point in time.
- The commit knows exactly how all of your files and directories looked and can therefore be used, e.g., to restore the project to that certain state.

Some Very Basic Cheat Sheet

- git config - -global user.email aghanhussain@gmail.com
- Git config - -global user.name “nasirhussain”

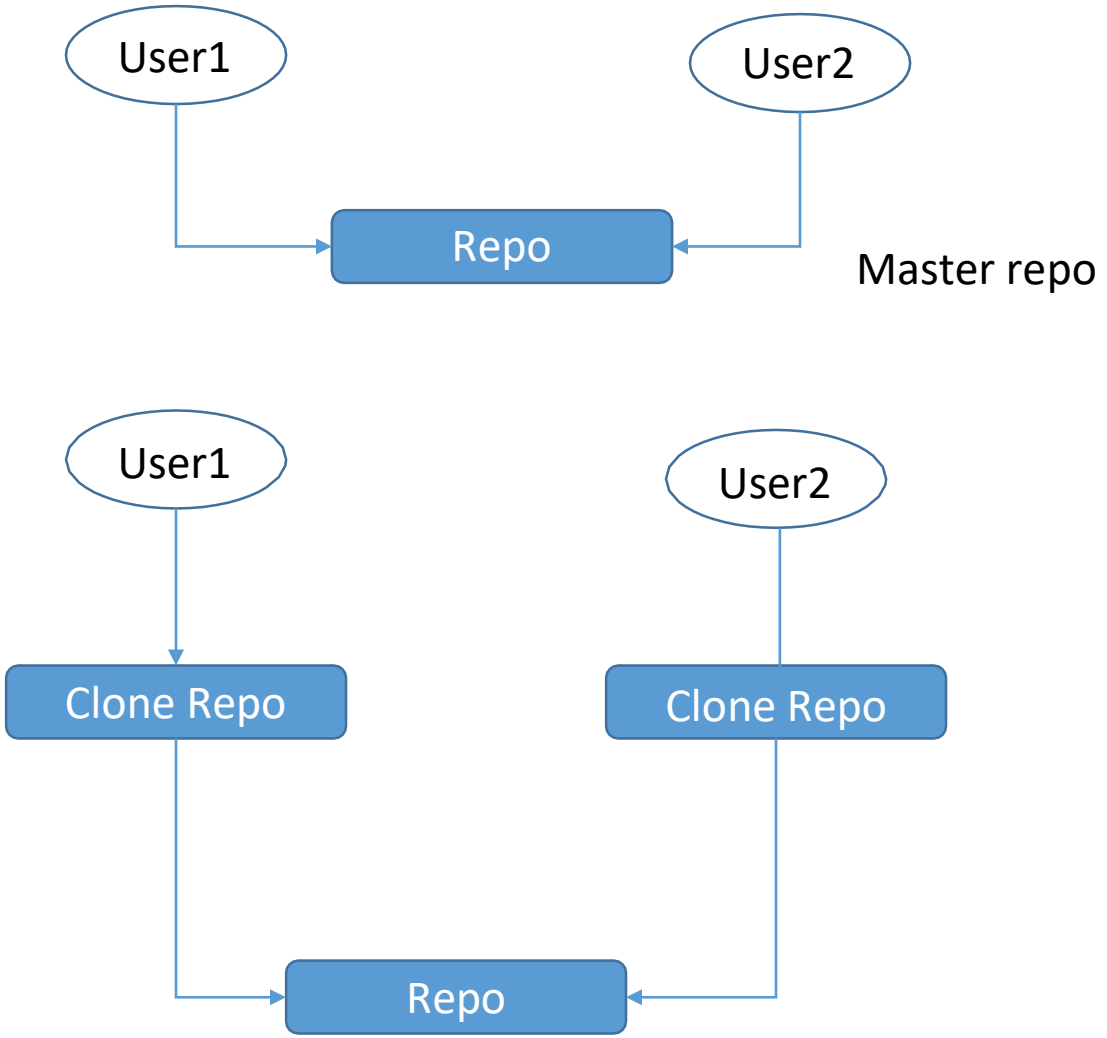
1. Git init
2. Git add [filename. Ext] or git add . Or git add –A
3. Git status
4. Git commit –m “message”
5. Git remote add origin [repository name]
6. Git push origin master
7. Git clone [repo name]
8. Git pull origin master


git init // a hidden folder will be created in root folder
User1
git status // untracked or tracked file will be displayed
Staging Area
git add[filename.txt] // to add file one by one
git add -A // to add all files together
User2
Git pull origin master // pull the changes
Snapshot
git commit –m “your message” // a snapshot of your commit will be created at your local repository
Git remote add origin [repoName // creates a link between local and cloud repository]
GITHUB
git push origin master // push the changes to cloud
Git clone [repo name]
// will clone the repo in same directory


File1
Snapshot1
Git commit –m “file1 added”
File1
File2
Snapshot2
Git commit –m “file2 added”
File1
File
File1
Snapshot3
Git commit –m “file3 added”

Branching & Merging

- Branching is a process in which a repository is cloned for every developer coding in the project so that their changes on same time may not intersect.
- Merging is a process in which both the clone repositories are merged and added to original so that their changes may reflect accordingly.



To check list of branch:

```
>>>git branch
```

Creating a new branch:

```
>>> git branch newBranch
```

Switch between branches:

```
>>> git checkout newBranch
```

Merging branches:
Load master branch first then

```
>>> git merge newBranch
```

Deleting braches:

```
>>>git branch –d newBranch
```