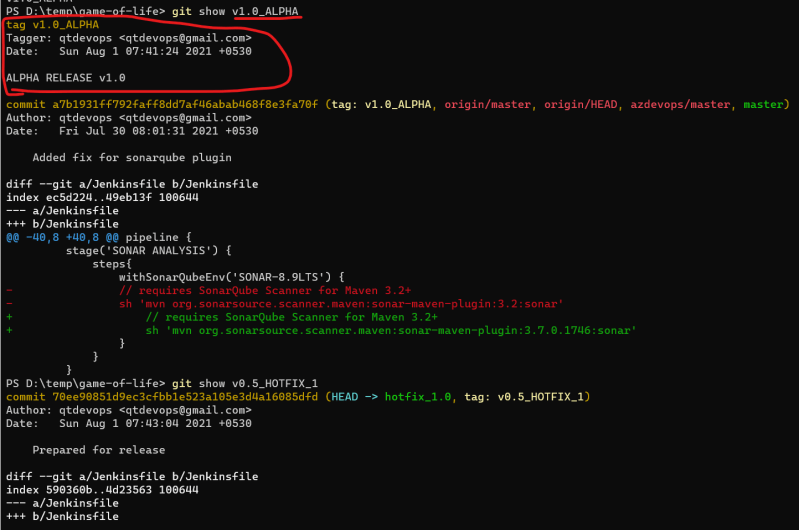
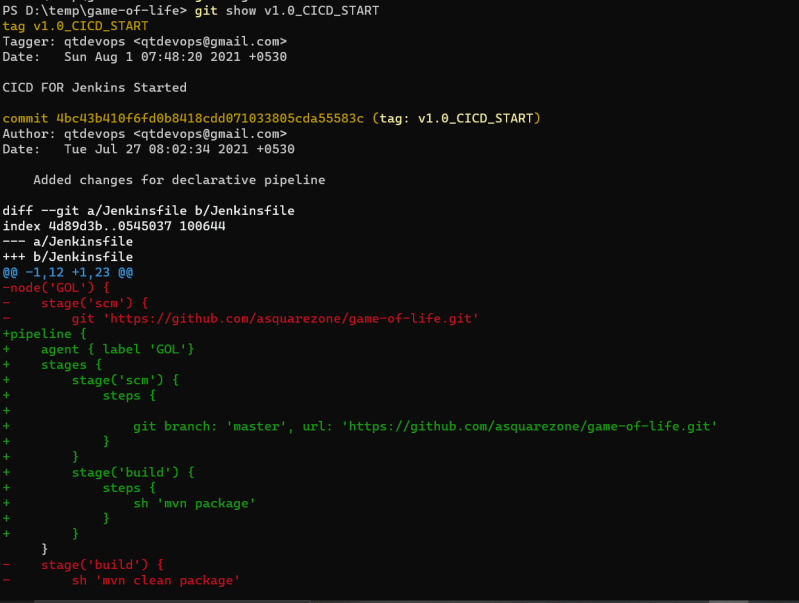
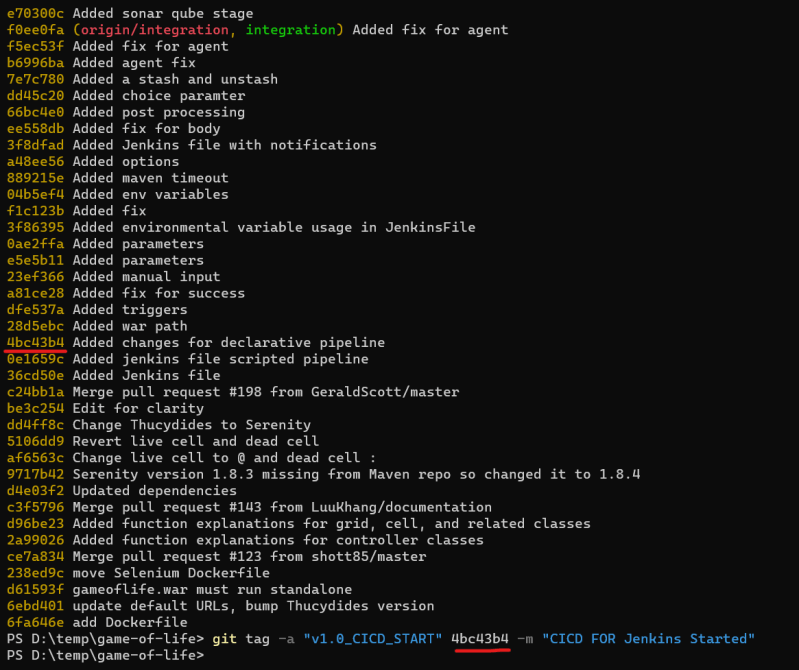
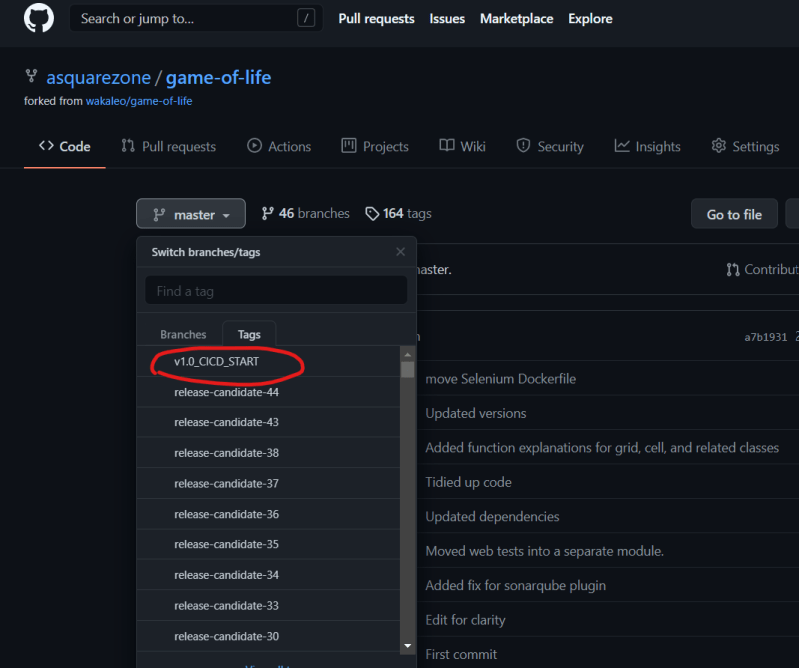
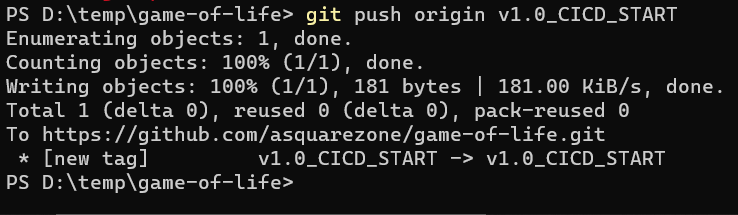
**Git Tag**

* Tags are ref’s that point to specific points(commits) in the Gi History.
* Tagging is generally used to capture a point in history i.e. used for marked version release.
* A tag is like a branch that doesn’t change, Unlike brnaches tags after being created have no further history of commits
* Creating a tag:
  + We can create two types of tags:
    - annotated: They are store as complete objects in Git database (local repo). They are checksum, require a message and store other important data such as name, email and date
  + git tag -a "<tag-name>" -m "<message>"
    - lightweight: They don’t require a message and they don’t store other data rather they are just a pointer to specific commit
  + git tag "<tag-name>"
  + To view Tags data use git show <tag-name> 

Tag older commits 

Pushing Tags git push <remote-name> <tag-name> 

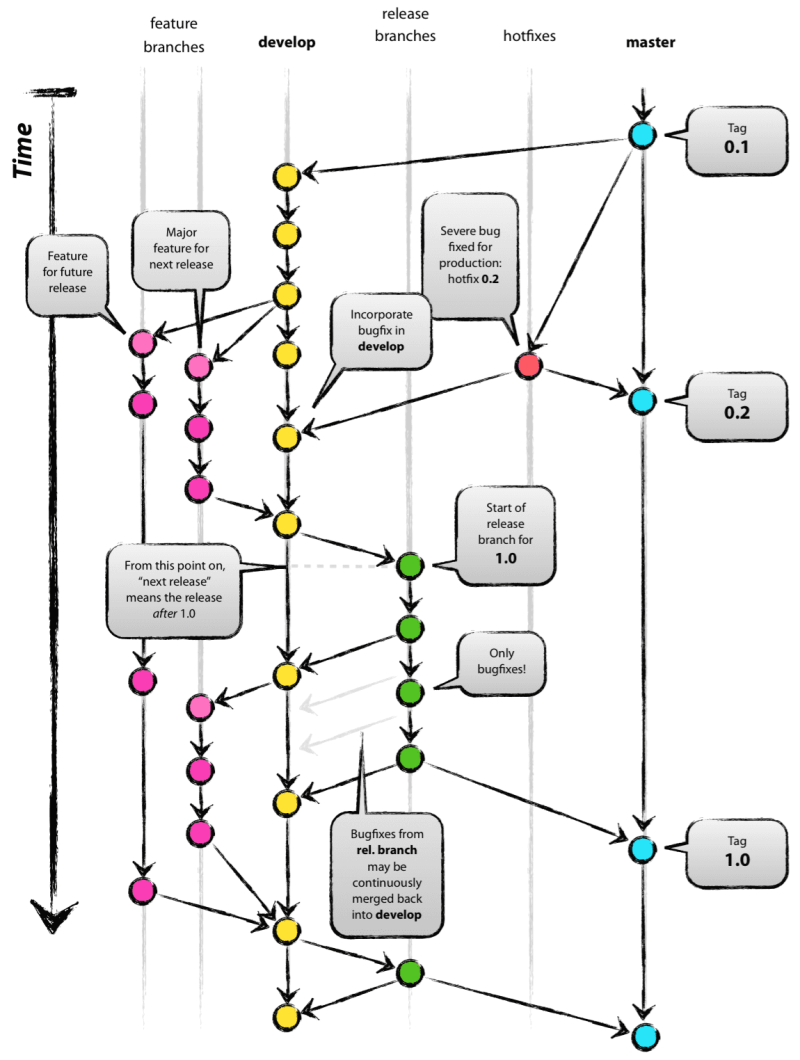
* + To push multiple tags at once use –tags
  + To delete the tag git tag --delete <tagname>

**Day Builds and Night Builds**

* Day Builds refer to builds done when developers are working and day builds should have ci/cd pipeline which gives feedback to the developer whether their commit is working correctly or not
* Night Builds refer to combination of all the work done by your dev team and this build will be taken by QA to perform Testing. We have only one Night Build per day. On the Night build we tend to run all the tests (unit, integration, load) so QA team will have confidence to test further.

**Branching Strategy**

* In this session we will be learning one braching strategy referred as GitFlow [Refer Here](https://directdevops.blog/2019/01/04/best-branching-strategy-git-flow/)



**Git Stash**

* Git stash temporarily shelves or stashes changes you have made in the working tree, so that you can work on something else
* To Stash your work: After you have made changes

git stash

* Re applying your stashed changes

git stash apply # apply the changes but the changes will still be present in stash list

git stash pop # apply the changes and removes the stashed changes from stash list

* Git stash will stash
  + Changes that have beend added to staged area, changes made to files that are currently tracked.
* Git stash will not stash
  + new files (untracked) in the working copy and files that have been ignored.
* When we have multiple stash items we can apply/pop any item from stash list

git stash pop --index n

* If you want to drop a stash git stash drop

**Git Bare Repo**

* Generally the Remote Repositories are hosted on the servers where they donot need working tree. All the changes to the Remote Repostiories will be pushed
* Git has a way of cloning git repository where only .git will be available this way of cloning is referred as bare clone.

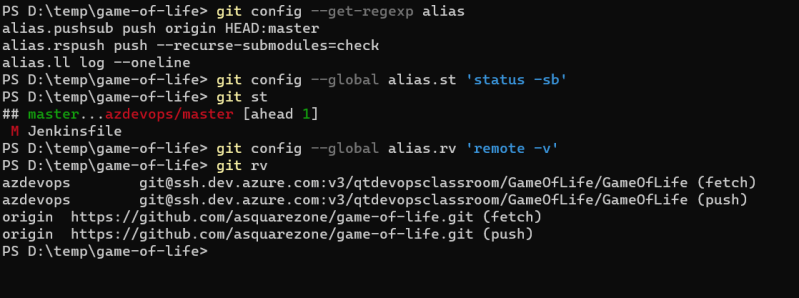
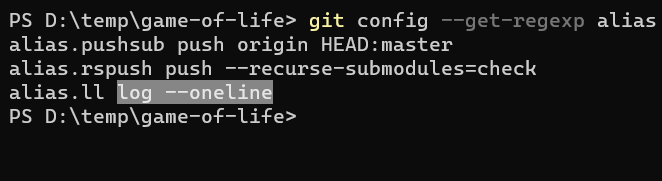
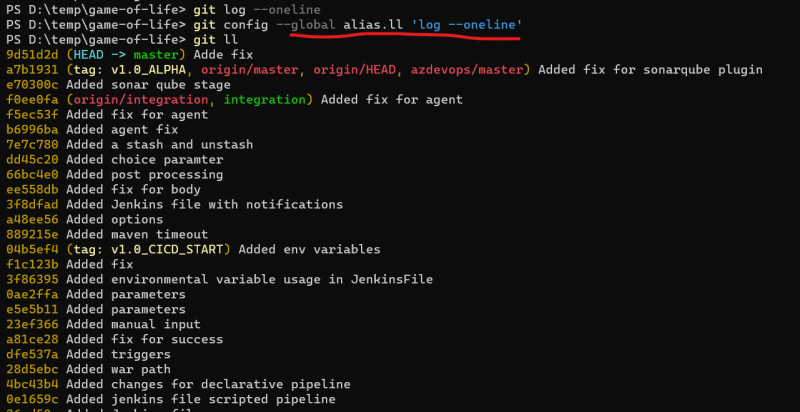
git clone --bare <git url>

**Git Mirror Repo**

* When we want to have backup of the repository where all the extended references to remot repository are also made available

git clone --mirror <git url>

**Git Aliases**

* Git gives us a capabiliyt to create alias commands so that we can save typing the whole command 

**Git Submodules**

* If you want to have a git repository with in other git repository(example you have 2 git repo for frontend, backend and middleware. And all together work for a project.)

**Git Hooks**

* Git Hooks are scripts that run automatically every time a particular event occurs in git repository
* In .git folder we have hooks folder
* Local Hooks
  + pre-commit
  + prepare-commit-msg
  + commit-msg
  + post-commit
  + post-checkout
  + pre-rebase
* Server-Side Hooks
  + pre-recieve
  + update
  + post-recieve
* When we have our own git server and then we can use hooks to add some kind of behavior or scripts to certain

**Web Hooks**

* For the cloud based git repositories like github/bitbucket etc we cannot control by writing Server Side hooks using .git\hooks folder so these cloud providers offer Web Hooks 