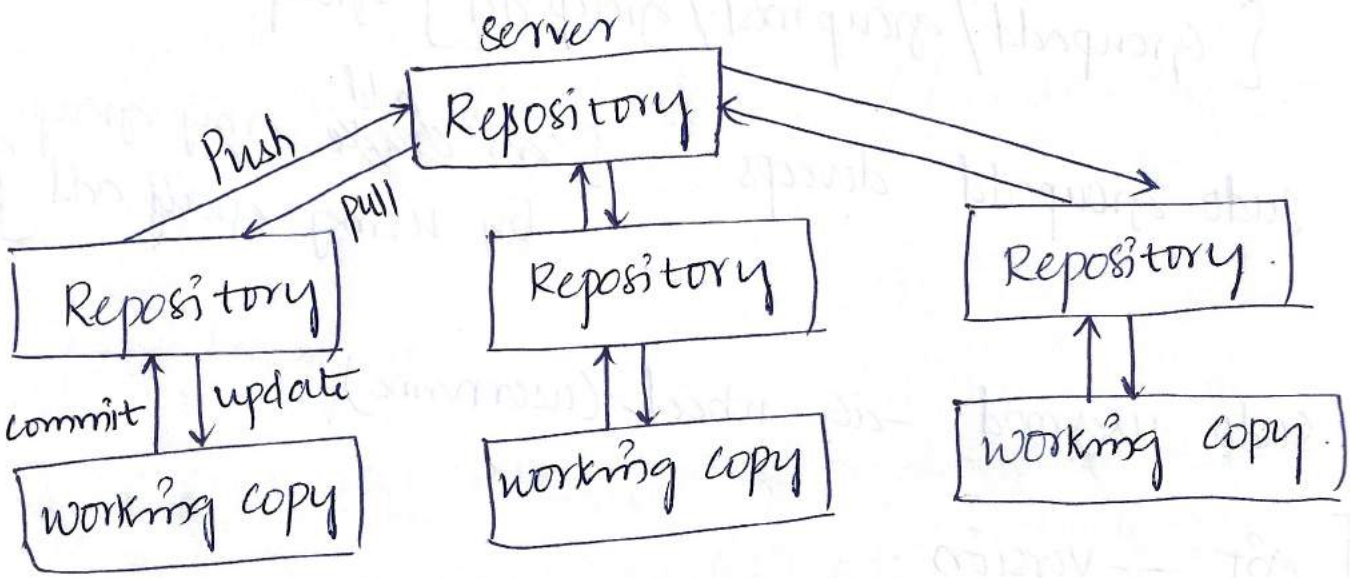
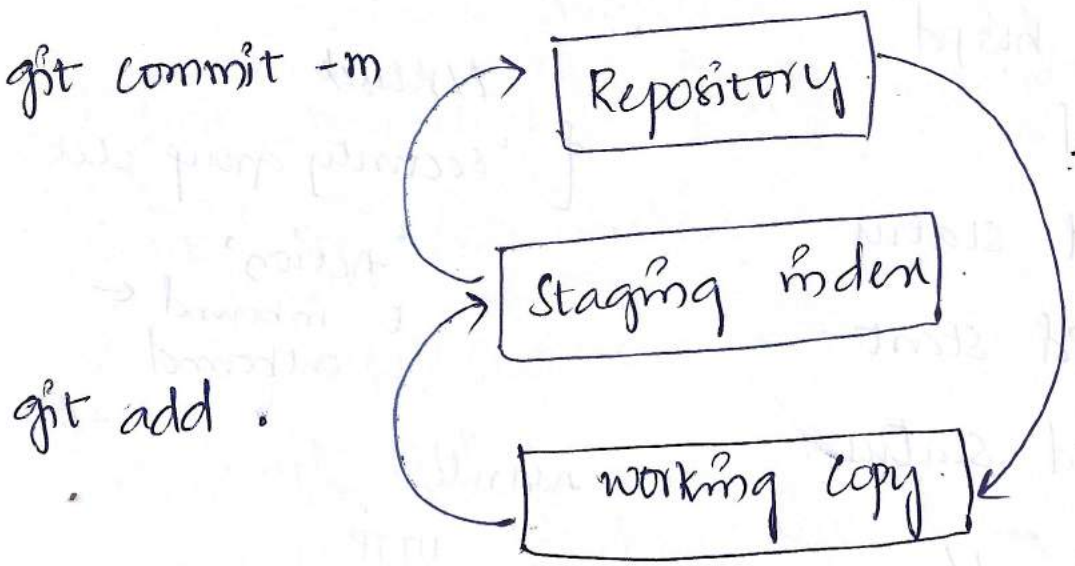
Version control tool:-

GIT :- it is Distributed Model

git is Distributed version control system.



GIT ARCHITECTURE :-



Code share -ability b/w the teammates & to manage History.

i) GIT (Local)

ii) GITHUB (Central)

iii) GITHUB ENTERPRISE (Rarely used). Paid one.

Sudo -i {root user}.

whoami

Yum install git

45) git --version.

mkdir git-repo.

ls.

→ cd git-repo/.

Pwd.

git init → (to convert to repo) ← to install

Pwd.

ls ⟹ when git installed it created a hidden directory called .git/

ls -a.

cd .git/

ls

cd ..

46) git status.

touch sample.txt.

ls -a.

cd .git/

ls

cd ..

git status). (to know the status).

47) git add . (. current location).

ls .git/

→ (index staging area)

cat .git/index.

git status

git branch.

→ (NO branch because no commit is made).

48) git commit -m "created sample.txt"

49) git branch.

→ (Master branch is created).

ls.

git status.

ls.

vi sample.txt

→ Added some text ←

git add.

git status

git commit -m "three lines added"

0) git log ⟶ (TO see commit id's and all the commit history).

git config --global user.name "username"

git config --global user.email "user email"

ls

vi sample.txt

⟶ Add text ⟵

git add.

git commit -m "message" } git commit -am "message"

git status

git log.

GIT HUB :-

Working with GITHUB

Github.com → Sign up
→ required details.

Create a repository.

Owner / Repository

github-repo. → centralized repository.

Created a repo.

→ Now take centralized repo data to local repo.

Go to account.

Open the repo. {ex:- github-repo}.

⇒ HTTPS process (copy the path).

→ Local work station ←

51) sudo -i

ls

51) git clone (path). (copy the path from GitHub).

ls

cd git-repo/ (Local repo).

52) git remote -v (if we get any path then it is a central repo).

cd ..
cd github-repo/

* git remote -v (if we check again we can see a path, hence it's a central repo).

⇒ instead reffering the whole path we can take origin. (Default).

touch login.java (placing some code)
ls
git status
git add.
git commit -m "login file created".
git branch
* main ⟶ (Branch main is created on GITHUB it is 'main' on Local (Git) it is 'master')

git log ⟶ (History is created).

git status
53) git push ⟶ (To push changes from Local to Remote (Central).

user name:-
Pass word:-
} (git GIT-HUB account details).

→ Access denied ← Pass-Ath is remove.

so, use token from GITHUB.

Go to Github.
settings.

* Dercloper setting.
* Personal access tokens. → Generate new token.

*Note
github-token.

Expiration
90 days.

* Generate the token.

*

- in HTTPS,
copy the token otherwise if we loose the token we have re-generate the tokens.

Go to local :-

user name : GITHUB username
Password : Token.

git log.

GO TO Github :- Check the github. Everything is updated (Branch).

git pull ⟶ (To confirm both local and central are in sync).
or to synchronize).

Vi login.java ⟶ (Add comment).

cat login.java

git add .

git commit -m "message"

git log.

cat login.java

git push

User name:

Password : Token.

54) git pull = Fetch + merge.

SSH Mechanism :- (GITHUB)

in SSH :- No username :-
and No password :- (i.e :- TOken)

SSH :- connecting to remote server securely.
The process is on the control machine you need a user and for user you have to generate Public and private key and on the Remote machine to the user you have to copy the public key.

or

on the remote side if you have the public key of user you can take the private key then straight forward with private key we can connect.

```
cd
ssh-key          (Create a Private & public key)

Enter
Enter
Enter
cd .ssh/
ls
cat id_rsa.pub   (copy this public key & paste in GitHub app)
```

→ [To copy the public key from Github account-

Github

settings.

* SSH and GPG Keys.

(create) - New SSH key.

ex:- root-key.] ←

mkdir ssh.

ls.

cd ssh/

ls.

git clone (paste) (SSH (options) copy the path).

ls

cd github-repo/

git remote -v.

ls.

touch readme.txt.

ls

git status

git add .

git commit -m "readme file added"

git push.

git log.

touch mytack.txt

ls

vi mytack.txt (Add (or) Edit comment)

git status

55) touch .gitignore (will ignore the content).

ls

ls -a

vi .gitignore

→ (add file for which content you want to ignore).

git status

git add .

git commit -m "message"

git Push

56) git diff (compare the changes (b/w branches, commits, local and remote)

57) Ps -ef (what are the services are running

Ps -ef | grep httpd (to know a particular service running (or) not.

Ps -ef | grep (service name)

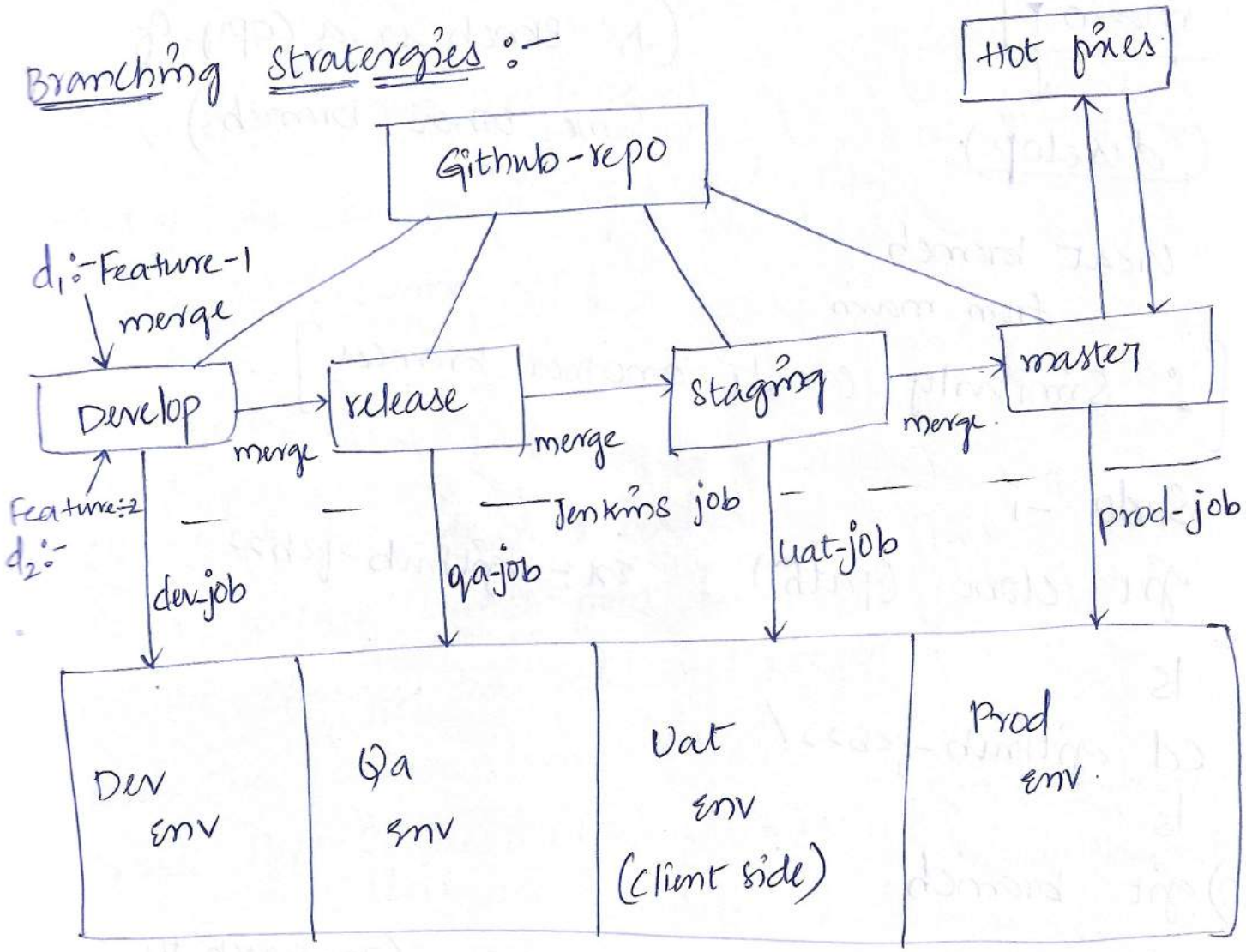
58) netstat -nltp (network ports running).

"To modify the commit message"

59) git commit --amend -m "message"

(it will also create a New commit id)

Branching Strategies :-



Creating a branch :- (from 'Github' side)

Create a New repo

github-feb22

☑ Add a Readme file

main

develop

(A Branch is a copy of some other branch).

creat branch from main

[∴ similarly create another branches].

sudo -i

git clone (path) ex:- github-feb22

ls

cd github-feb22/

ls

60) git branch.

61) git checkout (branch name) (To switch to particular branch)

Creating branch on Local :- (feature)

Make Sure you need to be on the source branch when creating a branch.

62) git branch (branch name) (creating a branch locally it a copy of source branch).

git diff main feature-1

→ if we see ~~any~~ -no- result then they are in same copy.

*Developer :-

1. clone/pull the code into local
2. create feature branch from develop/master
3. DO the changes then add and commit
4. Push the feature branch to GitHub.
5. create Pull Request (PR) (Merging feature branch to develop branch)

Reviewer :-

6. Recieve email with Pull Request number.
7. Review the code and provide the comments.
8. Merge the Pull request.
9. Delete the feature branch.

On Remote :-

settings

Branches.

Default branch :- Rename (or) change the branch to default branch

Branch Protection rules :- We can apply Protection rules and Pull request

**
Always main is default branch, but to change any other branch to default (Go to default branch)

[When we run (git push (or) git pull) (connect to the repo and sync default branch)

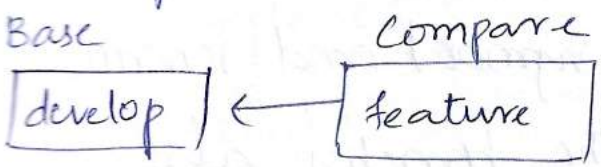
On Local :-

When we are not dealing with default branch we have give extra arguments, ex :- origin "origin is reference name".

ex :- git push origin feature-1

On remote :-

Compare and Pull request



→ cut branch = creating a branch from source.

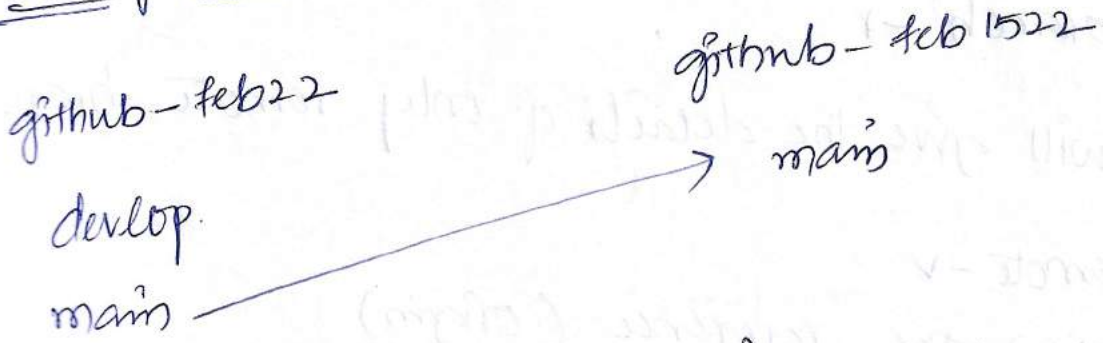
On Local

Run :- git pull origin develop to be in sync with develop on the remote.

Merging Locally :-

When ever you want to merge branch locally, make sure, you are on the target branch. then apply command git branch (source name).

Moving code from one repo to another repo :- (FORK).



→ git clone (path of another repo) ex :- feb1522

→ cd github-feb22
→ git remote add feb1522 (path copied).
→ git push feb1522 main

Rename - Branches :-

(Remote) :-

Go to branches and pencil symbol and rename

likewise we can delete the branches also.

On Local :-

cd github - feb22 /

git branch.

63) git branch -m (branch name) (New name).

ex :- git branch -m feature-1 sprint-1

→ To change the name of the branch on local

64) git branch -a

→ it will give the details of both local and remote branches.

65) git branch -r

→ it will give the details of only remote branches.

git remote -v

→ To view the reference (origin)

66) git remote rm (ref name)

→ To remove the reference name (feb1522)

mkdir git-merge-repo

cd git-merge-repo/ → git init

touch sample.txt →

git add .

git commit -m "sample file".

git status

git branch

ls

git branch develop

git branch

git branch feature-1

git branch

ls

git checkout feature-1

Vi sample.txt

→ Add (or) Edit comment.

git add.

git commit -m "message"

git diff master develop

git diff master feature-1

git checkout develop

→ When merging locally you should be on the target branch

git branch

git merge feature-1

ls

Cat sample.txt

git branch.

git diff develop feature-1

git branch.

***

git branch --merged (branch name).

→ To know the branches that are merged.

ex:- git branch --merged master.

***

git branch --no-merged (branch name).

→ To know the branches that are not merged.

ex:- git branch --no-merged master.

git branch --merged develop.

git branch.

→ conflicts :-

Can be resolved in 4 following ways :-

* Manual
* UI
* Tools
* IDE (Developer side)

ls

git branch

cat Sample.txt

git checkout master

git branch

ls

cat Sample.txt

Vi Sample.txt

→ edit : ex :- These changes are from master

git status

git add .

git commit -m "message"

cat Sample.txt

git diff master develop

git branch

git checkout master

git branch

git merge develop

→ conflict araised ←

cat sample.txt

→ it will show the changes ←

Vi sample.txt

→ delete 'dd' section first the select the appropriate ← :wq

git status

git add.

git commit -m "conflict fixed"

git status

git merge develop.

→ Already up to date ←

* This is Locally *

using VI :- (Remote).

(Local)

cd ..

rm -rf github-feb22

git clone (path).

ls

cd github-feb22

git branch.

git checkout release.

git branch.

git branch release-1

git checkout release-1

git branch.

vi sample.txt

→ edit ex:- These changes are from release-1

git add.

git commit -m "release-1 br changes"

→ on Remote

release

1. This is first line.
2. This is second line.
3. These changes are from release branch.

Local :-

git push origin release-1

On Remote :-

Pull request

Base: release ← Compare: release-1

target. source

{ in local 1-1 git merge. in remote 1-1 git pull }

create pull request

Resolve conflict

commit merge

mark resolved

1
2
3 <<<<<
4
5 ====
6 >>>>>

} select delete.

cd..
ls
cd git-merge-repo/
ls
git branch
cat Sample.txt
git checkout develop
cat Sample.txt

Local	UI
Only target branch changes the same	Both the Source & target will change

Using Merge Tools :- (Cold process).

```
git branch
git branch sprint-1
git diff develop sprint-1
ls
vi sample.txt
```

→ 1
2. These changes are from develop br

```
git add.
git commit -m "message"
git branch
git checkout sprint-1
git branch
vi sample.txt
```

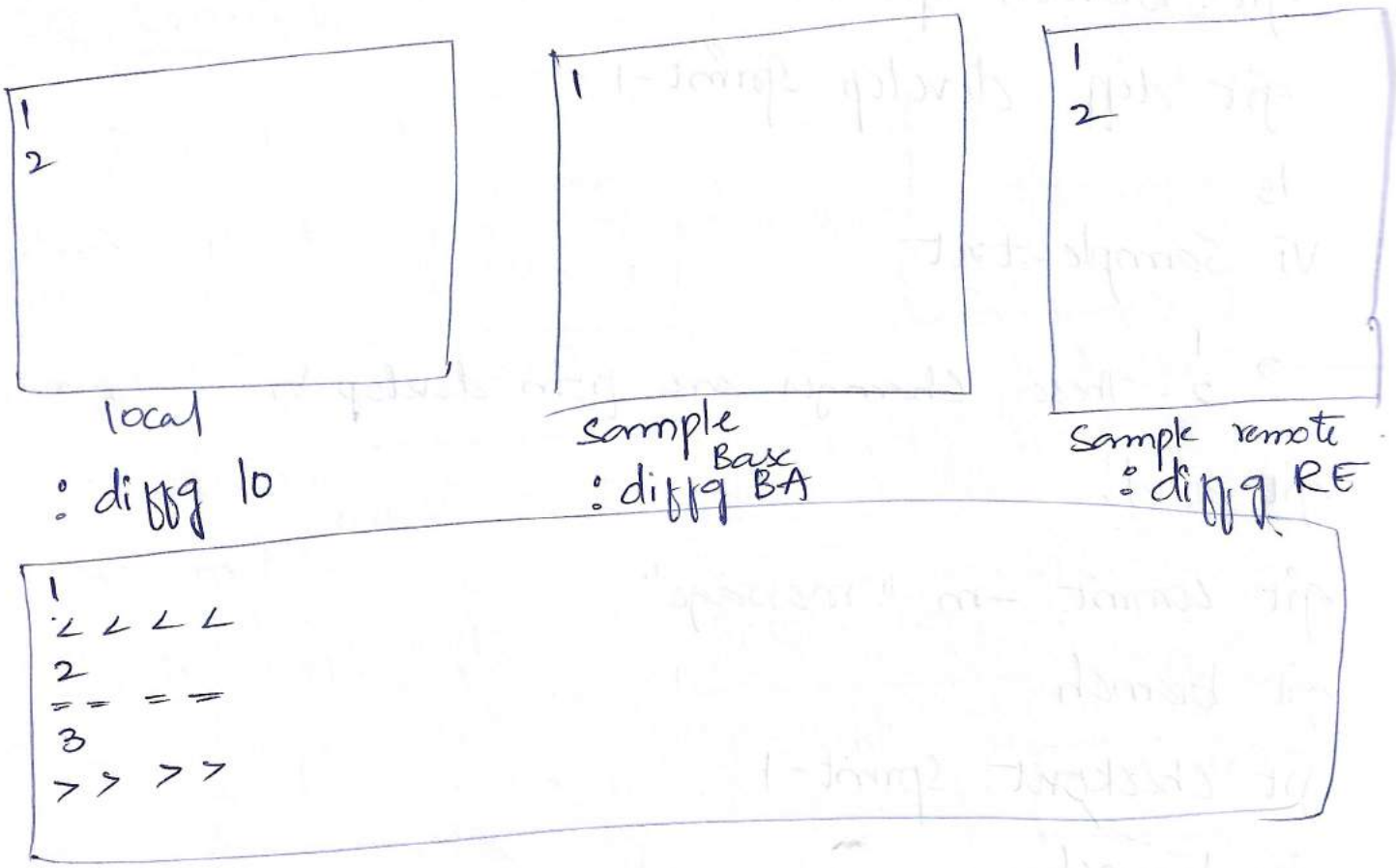
→ 1
2. These changes are from sprint-1 br

```
git add.
git commit -m "sprint-1 changes"
git branch
git checkout develop
git branch
git merge sprint-1
```

git merge tool

{ git merge tool
ex :- (vimdiff) }

Enter.



conflict

: wqa

git add .

git commit -m "conflict resolved"

git merge sprint-1

Branch Deletions :-

Remote :-

Go to Branch

Delete the branch which is not needed

ex :- feature-1

release-1

Locally :-

67) git branch -d sprint-1 (soft)

→ branch deleted

git branch -D (Branch name) (Forceful).

68) git branch --merged | grep -v * | xargs git branch -D

→ To delete branch which are already merged.

69) git push origin :main -1

→ To delete remote branch from local server.

Versioning :-

```
ls
git log
git branch
git log
rm -rf *
ls
mkdir git-repo
cd git-repo/
git init
touch sample.txt
vi sample.txt  ⟶ (Edit :- This is first line).

git add .
git commit -m "message"
git branch
git log
echo "This is the second line" >> sample.txt
cat sample.txt
git commit -am "message"
git log
git branch develop
```

git branch.

git checkout develop.

git branch.

ls

git log.

echo "This is the third line" >> sample.txt.

git commit -am "message"

cat sample.txt

git log.

git checkout master.

git log.

git show (commit id).

→ we can see actual data in the commit.

git diff (commitid) (commit id-2)

→ To see the changes in history b/w commit id to latest id).

git checkout develop.

git branch

git log.

To Go To particular commit Id :-

70) git checkout (commit id)

→ To switch to previous (or) particular commit Id we can use checkout command.

71) git switch -c (new branch name)

```
git branch
ls
git log
ls
cat sample.txt
```

Tagging (Alias of commit id).

i) light weighted Tag

ii) Annonated Tag.

72) git tag -L

→ To check if any Tags are already present.

```
git tag V1.2
```

→ If provided no commit id then the tag will apply to the latest commit Id

git log. (Now you can see a tag).

git show (version). →

→ To see the particular version and its history.

73) git tag (version) (commit id).

→ To give give version to a particular commit id.

git diff (version 1) (version 2)

→ To see the difference b/w two versions.

git checkout (version)

→ To switch to the version.

git switch -c (version).

74) git tag

→ To see versions created sofar.

***

75) git push origin --tags

→ To push all the tags to remote.

git push origin (version)

→ To push the particular tag to remote.

Annotated Tags:-

→ To add a message to tags.

76) git tag -a (version) -m "message"

→ To add message to particular tag.

77) git tag -d (version)

→ To delete the version.

78) git push origin :(version)

→ To delete the version from the local in the remote.

79) git ls -remote --tags

→ To know remote tags from local.

80) git tag -d $(git tag -l)

→ To delete all the tags from local.

81) git ls -remote --tags -refs origin

→ it will give whatever the remote tags present in github side

82) git ls -remote --tags -refs origin | cut -f2 | xargs git push origin --delete

→ (To delete all the tags) ———

Reverting changes :-

83) git restore (file name) (before (add .) command
* working copy *.

84) git restore --staged (filename)
→ (After add. command) * Staging index *.

git restore (file name) to go back to original
- No data -

After the dog is created to revert the changes command,
we have commit 'reset'

* git rest soft
* git rest mixed
* git rest Hard

85) git reset --soft (commit id) (git reset 'soft')
↓
For which commit we want to go.

→ Here still the data is present.
ex:- cat sample.txt
↓
→ From (repo to staging index, this data is present.

86) git reset --mixed (commit id) (git reset mixed)
↓
For which commit we want to go.

→ Here No data is present.
→ From repo to working copy

87) git reset --hard (commit Id) (git reset hard).

↓

For which commit Id we want to go

→ No data, will directly go to the commit id we provided

88) git revert (commit Id)

→ It won't delete the commit Id instead creates new commit on top of existing one.

Cherry-Pick :-

* Whenever you want to take a particular commit (Merge) into your target branch "cherry pick" is used.

*89) git cherry-pick (commit Id).

Stashing :- (This works after the commit -m)

→ Storing the data in buffer.

90) git stash list (TO view the list ex:- stash@{1})

91) git stash (to stash the changes)

92) git stash apply (to bring back changes)

93) git stash Pop (it will remove the stash entry)

94) git stash apply (Id).

→ if we want to work with particular stash.

ex:- git stash apply stash@{1}.

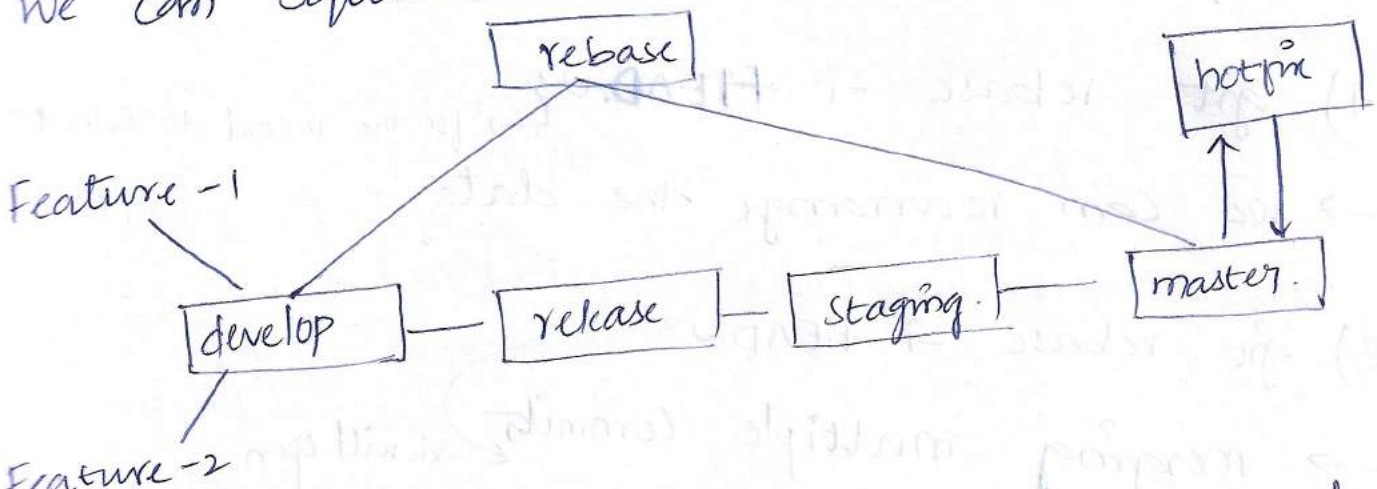
→ We can see stash Id's from stash list.

Rebase :- (Re-arrange the history).

To maintain proper history we will use 'Rebase'

We can shuffle the history.

We can Squeez the history.



Feature

a.txt
aaa
b.txt
bbb

master.

a.txt
b.txt
c.txt

Fast forward.

95) git log --online (short form of log id) and message.