Stash:-- Git stash is temporary storage area.

Often, when you've been working on part of your project, things are in a messy state and you want to switch branches for a bit to work on something else. The problem is, you don't want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is the git stash command.

Create some sample files

add those files from workspace to stagin/index area

check the git status command

git stash list

List the stash entries that you currently have. Each stash entry is listed with its name (e.g. stash@{0} is the latest entry, stash@{1} is the one before, etc.), the name of the branch that was current when the entry was made, and a short description of the commit the entry was based on.

git stash save "stash-1"

git stash list

git stash show -p stash@{0} : To check the what and all files in stash memory.

Show the changes recorded in the stash entry as a diff between the stashed contents and the commit back when the stash entry was first created. When no <stash> is given, it shows the latest one. By default, the command shows the diffstat, but it will accept any format known to git diff (e.g., git stash show -p stash@{1} to view the second most recent entry in patch form). You can use stash.showStat and/or stash.showPatch config variables to change the default behavior.

git stash pop

Remove a single stashed state from the stash list and apply it on top of the current working tree state.

git status

git stash apply stash-name

Like pop, but do not remove the state from the stash list.

git stash drop stash-name

Remove a single stash entry from the list of stash entries.

GIT MERGE / REBASE:

It's simple, with rebase you say to use another branch as the new **base** for your work.
If you have for example a branch `master` and you create a branch to implement a new feature, say you name it `cool-feature`, of course the master branch is the base for your new feature.
Now at a certain point you want to add the new feature you implemented in the `master` branch. You could just switch to `master` and merge the `cool-feature` branch:

```
$git checkout master
$git merge cool-feature
```

but this way a new dummy commit is added, if you want to avoid spaghetti-history you can **rebase**:

```
$git checkout cool-feature
$git rebase master
```

and then merge it in `master`:

```
$git checkout master
$git merge cool-feature
```

This time, since the topic branch has the same commits of master plus the commits with the new feature, the merge will be just a fast-forward.