GIT

Creating a branch

git branch “branchName”

git checkout branchName

or

git checkout –b branchName

push the code using git push origin branchName

cherry-picking

* cherry picking in git means to choose a commit from one branch and apply it into another
* This is contrast with other ways such as merge and rebase which normally applies many commits into another branch.
* Make sure you are on the branch you want to apply the commit and then use below command.

Git cherry-pick <commit-hash>

Change code in the branch and commit it

Again change one more time and commit it,so that we can compare easily

Now push to codes to our branch git push

git log ---> it lists out all the commits happened till now

now we are going to apply one commit from branch on the master (we are going to add the first commit to master)

copy the commit id : eg: 4519b4713f4ea70dffc0c685e0d851a56a577550

checkout to master branch --> git checkout master

git merge 🡪 it applies/merges all the commits from develop to master

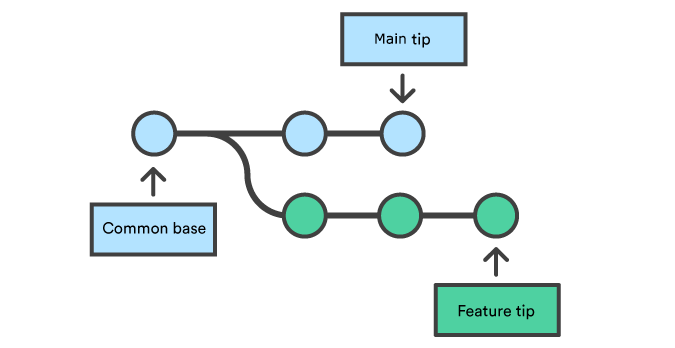
but we want to add specific commit on master branch🡪

git cherry-pick commit\_id

git push origin master

go to master and check now

Margin and Rebase

* The first thing to understand about git rebase is that it solves the same problem as git merge.
* Both of these commands are designed to integrate changes from one branch into another branch---they just do it in very different ways.
* 
* Consider what happens when you start working on a new feature in a dedicated branch, then another team member updates the master branch with new commits. This results in a committed history.
* Now,lets say that the new commits in master are relevant to the feature that you are working on.To incorporate the new commits into your featute branch, you have two options.

1. Merging

2. Rebasing

1.Merge:

Easiest option is to merge the master branch into feature branch

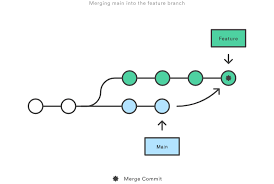
git checkout branchName

git merge master

or

git merge branchName master

This creates a new “merge commit” in the feature branch that ties together the histories of both branches



2.Rebase:

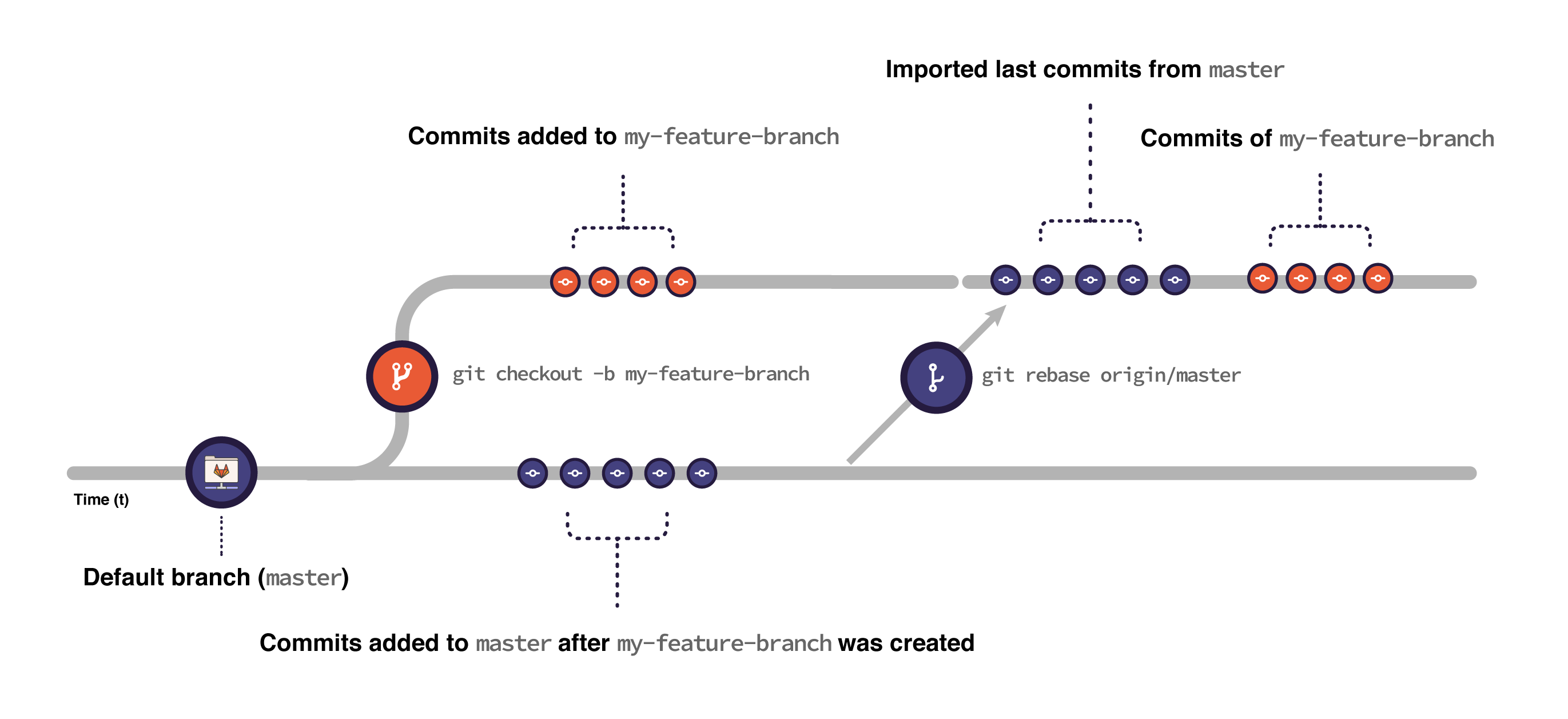
Alternative to merging,you can rebase the feature branch onto master branch.

git checkout branchName

git rebase master

This moves the entire feature branch to begin on the tip of the master branch, effectectively incorporating all of the new commits in master.But,instead of using a merge commit,rebasing re-writes the project history by creating brand new commits for each commit un the orginal branch.

The major benefit of rebasing is that YOU get a much cleaner project history.First,it eliminates the unnecessary merge commits required by git merge.second,as you can see the diagram,rebasing also results in a perfectly linear project history



Commands:

Create a branch feature

Git chechkout master

Git log –-oneline

Git checkout develop

Git log –oneline

git checkout feature

git log –oneline

do some changes in master and commit it and push it

git log –oneline

we will have the new commit

now checkout on develop branch and changes in code an commit it and push it

go to develop branch and git merge master

opens a window

press i

specify msg

press esc :wq and enter

check histories of two branches

git log –-oneline and master branch will be unaffected and in develop branch we get merge commit

if u do rebase this commit will not create

Rebase:

Checkout to feature branch and log it and change the code and push it

Check histories of master and feature

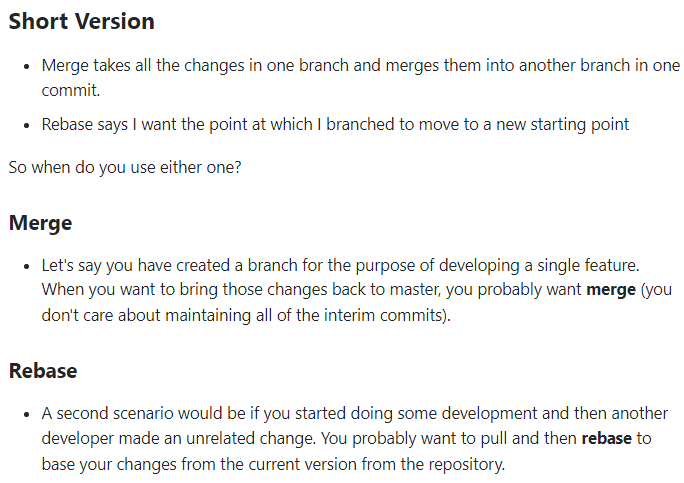
Git rebase master

Check histories of master and feature

Keypoints:

* Git merge apply all commits from branch A into branch B in one commit with final result
* Git merge doesnot rewite commit history,just adds one new commit
* Git rebase gets all commits from both branches and applies them one by one
* Git rebase rewrites commit history but doesnot create extra commit for merging

Which method to choose?



Git pull vs fetch

Git fetch only downloads lastest changes into the local repository.It downloads fresh changes that other developers have pushe to the remote repository since the last fetch and allow you to review and merge manually at a later time using git merge.Because it doesnot change your working directory or the staging area.it is entirely safe and you can often as you want.

Git pull downloads lastest changes into the local repository and it also automatically merges changes in your working directory.it doesnot give you a chance to review the changes before merging, and a s a consequence , merge conflicts can and do occur.

One important thing to keep in mind is that it will merge only into the current woking branch.Other branches will stay affected.

Git pull = git fetch+git merge

Go to master and add changes to file and push it and log the commits

Here we get merge conflicts

Git fetch

Git merge

Git revert

How do u revert a commit that has already been pushed and public?

Two approaches

1.fix the bad file and commit it,so that we can fix error

2.you can create a new commit that undo all changes that were made in the bad commit.

git revert <bad commit\_id>

git push origin master

make some changes in a branch and push it to that branch and revert it and push it by using above commands.

Git Stashing:

Why we need?

We did some modifications on one branch A and due other priorities requirement,we want to switch on to other branch B and our branch A is not staged and I want to comeback after some time to this branch A after I complete my work in branch B,then what ever changes we have done are put into a temp memory called stash.

Do some changes in one branch

Git status

Git stash—changes to be commited

Git stash list

Git status—no changes

Add another change and put in stash

We can also save the stash msg directly in stash by:

Git stash save “just saving in stash”

Change to other branch and come to this branch and apply these command to get the stashed changes

Git stash apply 🡪brings the lastest stashed item only

Git stash apply stash@{id} ->brings specific item

Even after removed from stashed memory,it is holding the stashed item reference

Git stash pop🡪latest item will be removed from stashed area and brings it into working area

Git stash drop 🡪 latest item will be removed from stashed area in first index

Git stash drop stash@{id}--> specific item will be removed from stashed area

Git stash clear ->clears all the data(all our changes wiil be deleted ,risky)

Git stash branch branchName🡪 Creating a branch for the stashed data also by:

git branch –D branch-name (delete from local)

git push origin :branch-name (delete from stash)

merge conflicts:

A merge conflict is **an event that takes place when Git is unable to automatically resolve differences in code between two commits**. Git can merge the changes automatically only if the commits are on different lines or branches.