the NeRd Talks

git rebase vs. git merge

Relatore: Giuseppe Sucameli

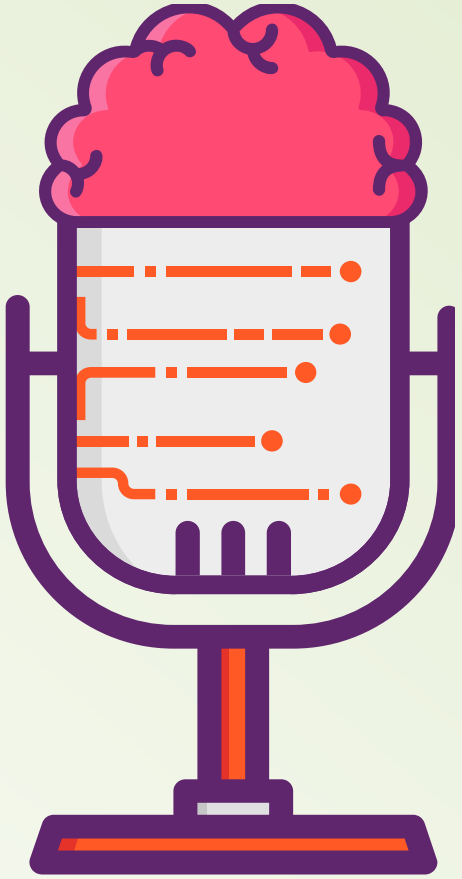
the NOT SO NeRd Talks

git rebase vs. git merge

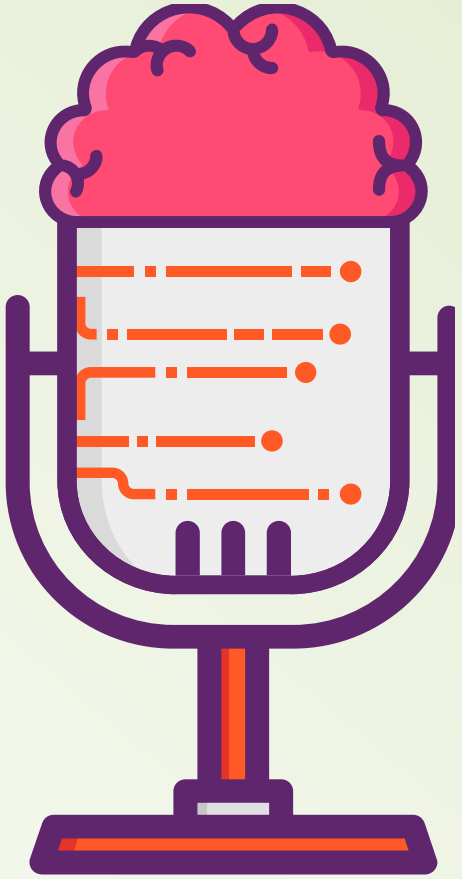
Relatore: Giuseppe Sucameli

git rebase vs. git merge

Relatore: Giuseppe Sucameli

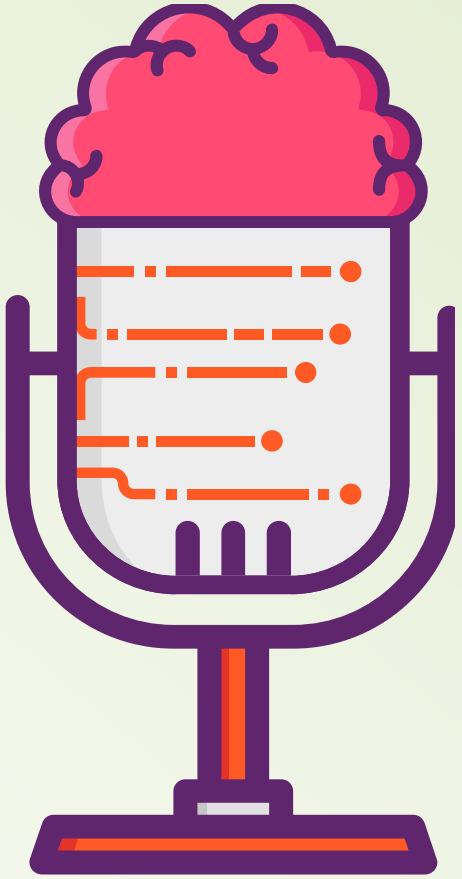


git rebase vs. git merge (link)



[git rebase vs. git merge](#) (link)

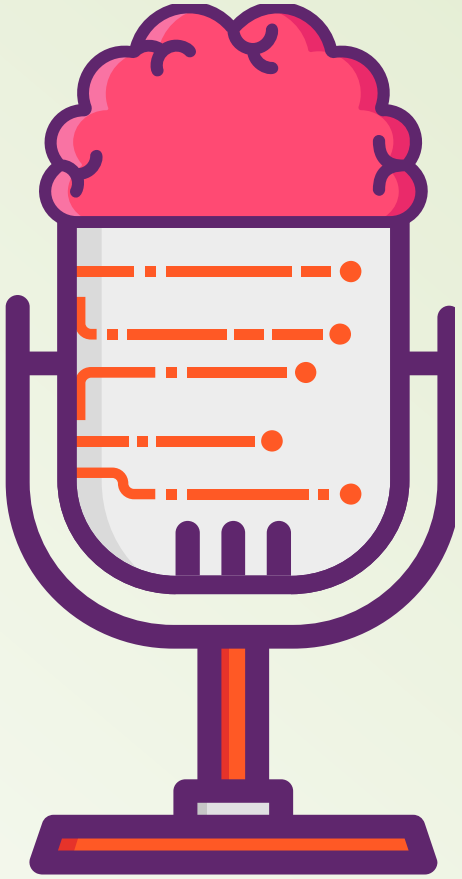
THE END



git rebase vs. git merge (link)

THE END

Now you also know what to do
for any other questions…

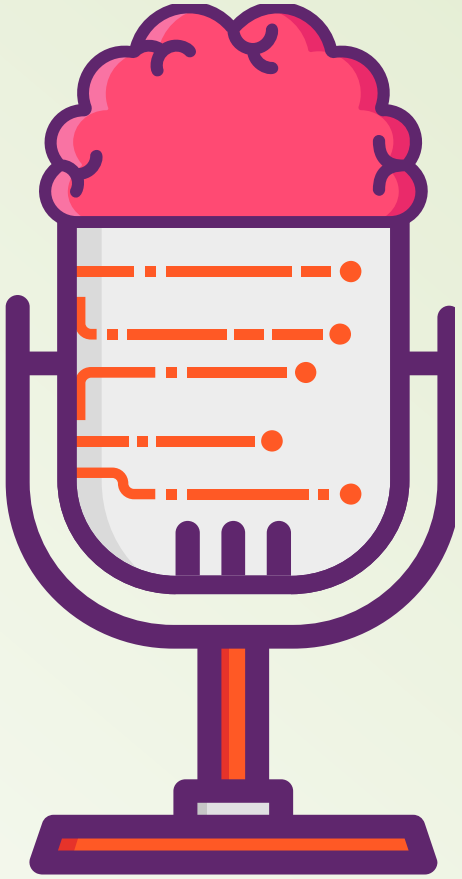


git rebase vs. git merge (link)

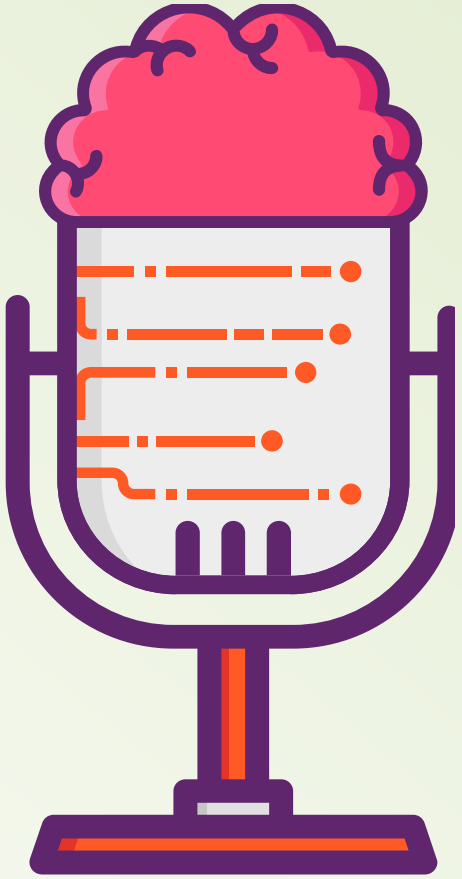
THE END

Now you also know what to do
for any other questions…

lmgtfy!



THE END?



THE END?

Not really :)

git rebase vs. git merge

Both commands are designed to integrate changes from one branch into another branch...

master

git rebase vs. git merge

Both commands are designed to integrate changes
from one branch into another branch...

... but they do it in very different ways.

master

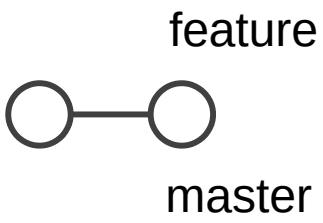
git rebase vs. git merge

After you create a ***feature*** branch
from ***master***...

master

git rebase vs. git merge

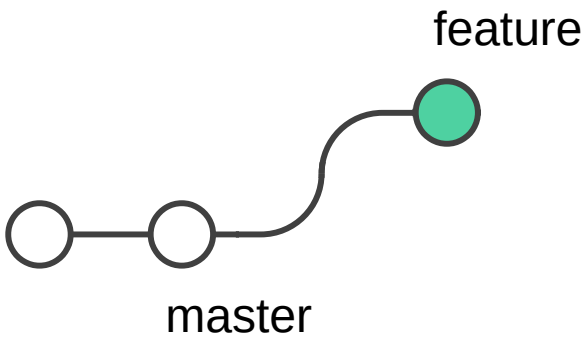
After you create a ***feature*** branch from ***master***...




```
git checkout -b feature master
```

git rebase vs. git merge

After you create a ***feature*** branch from ***master***...

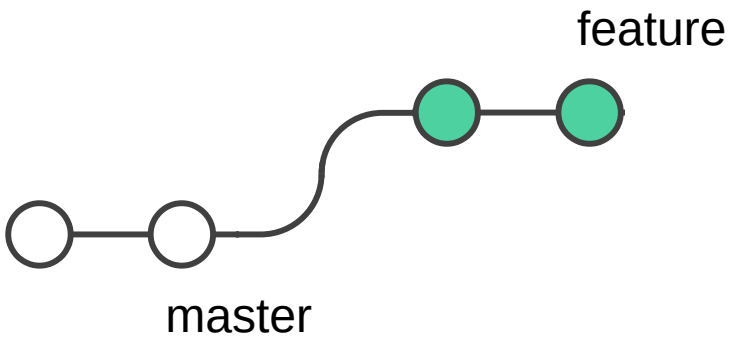


... and you work on it

```
git add <files>...
git commit -m "..."
```

git rebase vs. git merge

After you creates a ***feature*** branch
from ***master***...



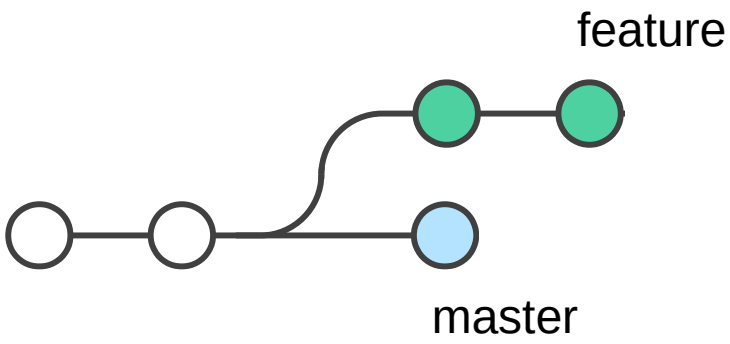
... and you work on it

adding few commits...

```
vi <file>
git commit -a
```

git rebase vs. git merge

After you creates a ***feature*** branch
from ***master***...



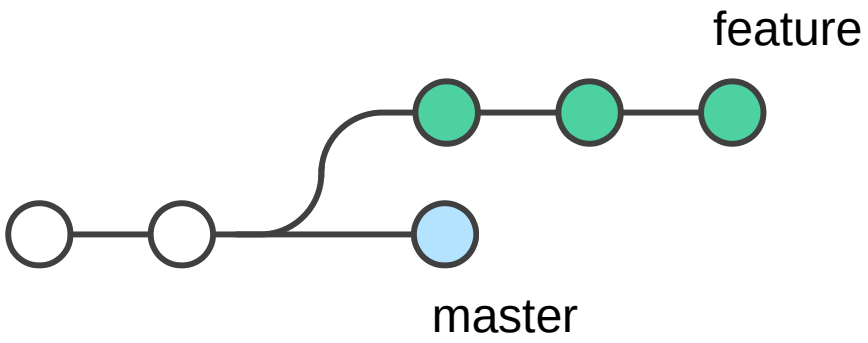
... and you work on it

adding few commits...

... someone else updates
master branch with new commits.

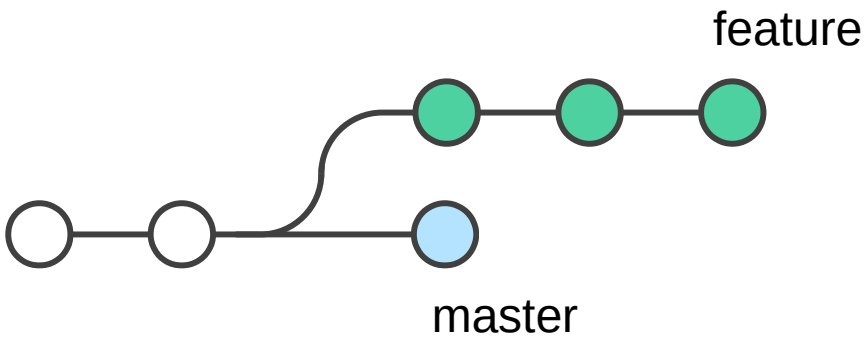
git rebase vs. git merge

Now, let’s consider that you need new changes
in ***master*** (upstream changes) to continue your work.



git rebase vs. git merge

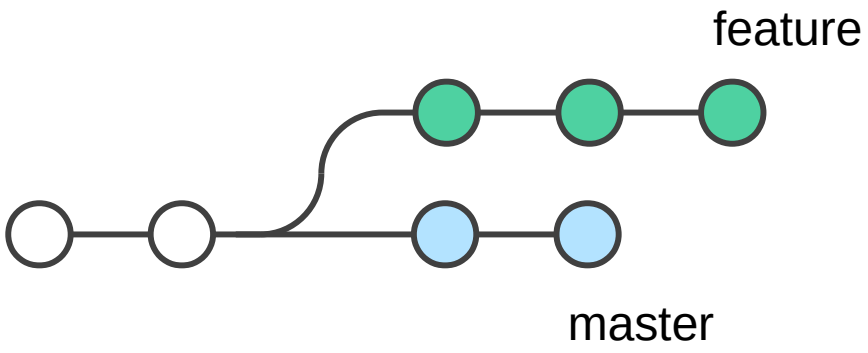
Now, let’s consider that you need new changes
in ***master*** (upstream changes) to continue your work.



That’s one of those moments
you have to choose how to proceed.

git merge

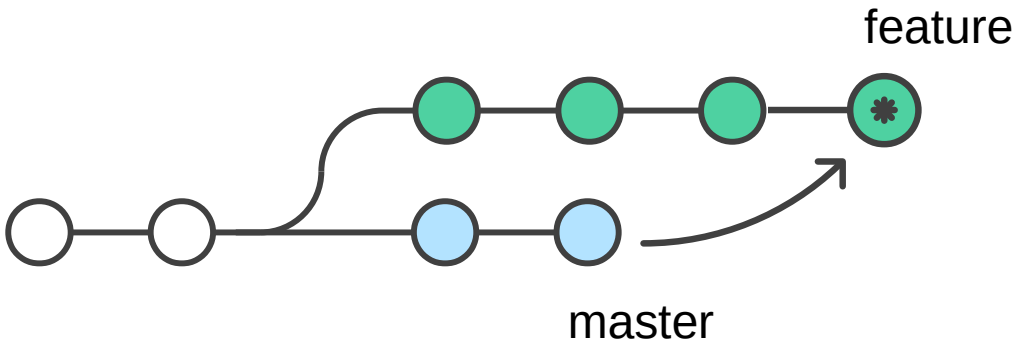
Merge is the easiest way to reintregrate changes from ***master*** branch into your ***feature*** branch.



git merge

Merge is the easiest way to reintregrate changes from ***master*** branch into your ***feature*** branch.

= merge commit

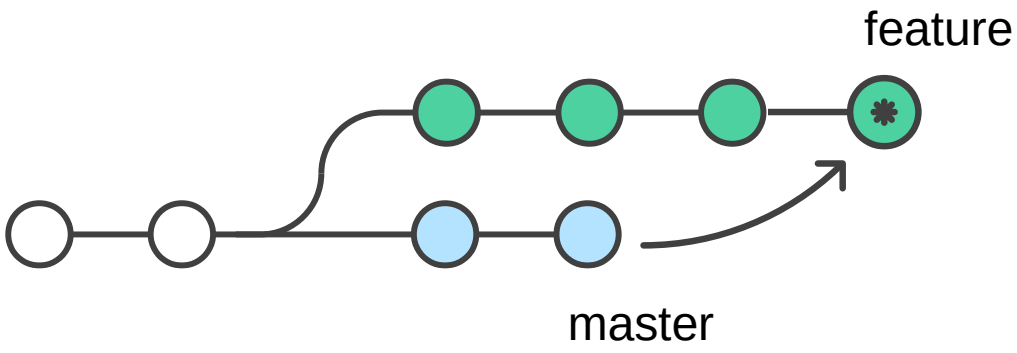



```
git checkout feature
git merge master
```

git merge

Merge is the easiest way to reintregrate changes from ***master*** branch into your ***feature*** branch.

= merge commit



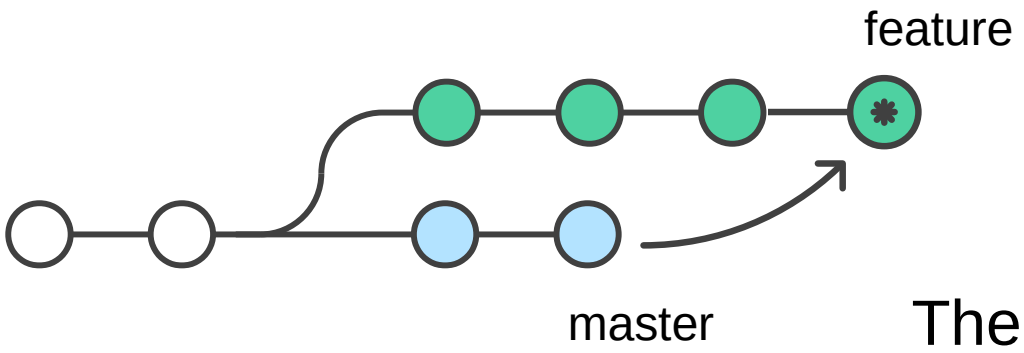
This creates a new **merge commit** in the ***feature*** branch

```
git checkout feature
git merge master
```

git merge

Merge is the easiest way to reintregrate changes from ***master*** branch into your ***feature*** branch.

= merge commit



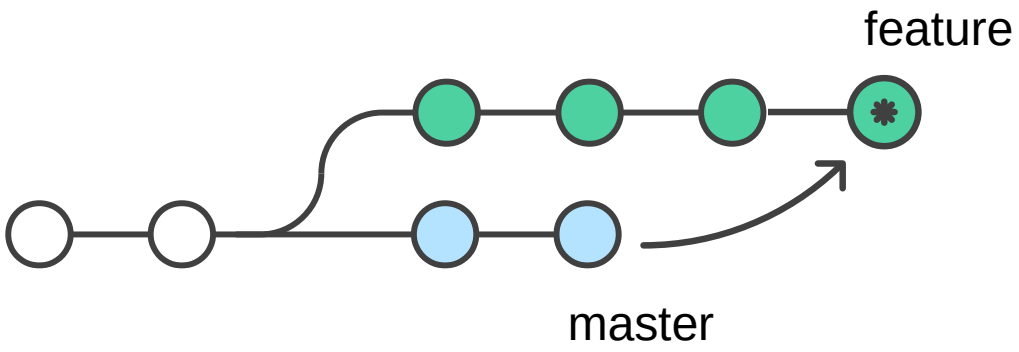
This creates a new **merge commit** in the ***feature*** branch

The **merge commit** keeps trace of the history of both ***master*** and ***feature*** branches

```
git checkout feature
git merge master
```

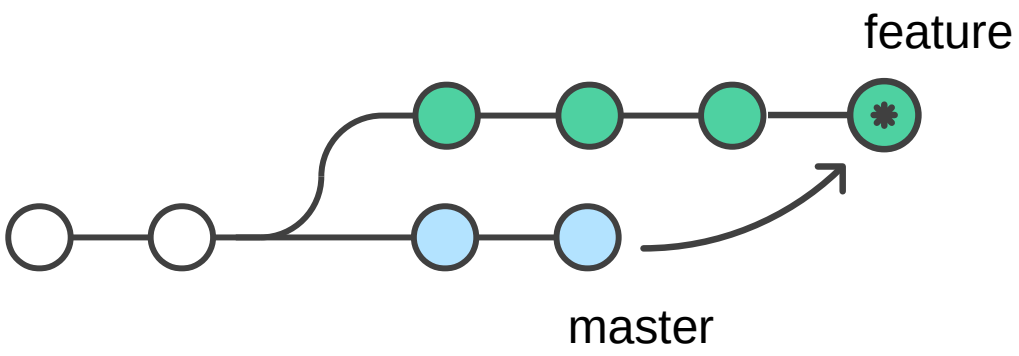
git merge

+ Merge operation is safe, i.e. *non-destructive*.



git merge

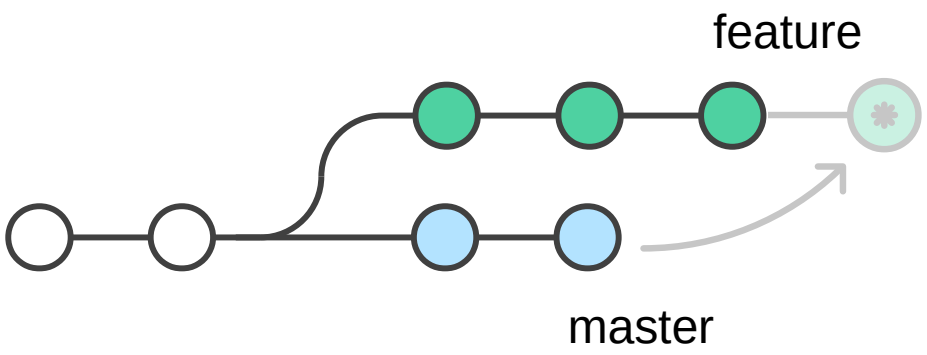
+ Merge operation is safe, i.e. *non-destructive*.



A merge operation can be easily undone...

git merge

+ Merge operation is safe, i.e. *non-destructive*.



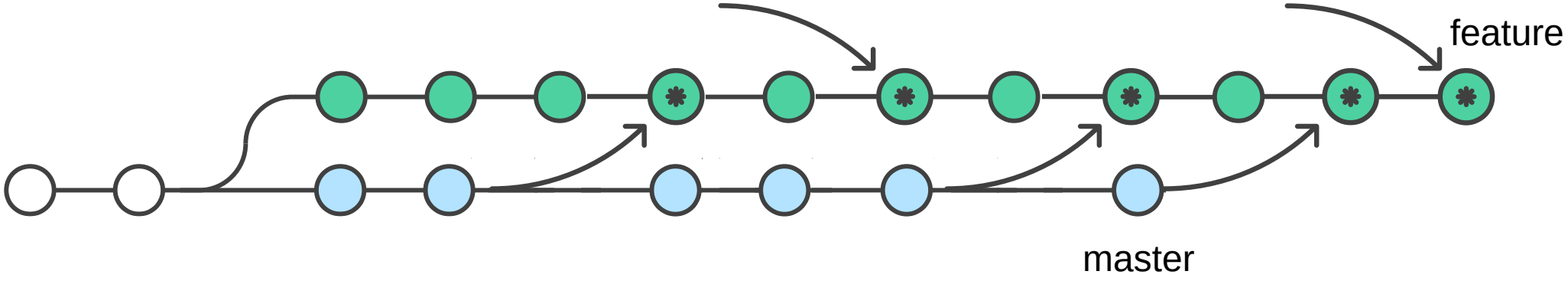
A merge operation can be easily undone...

... the same way of undoing any other commit.

```
git checkout feature
git reset --hard HEAD^
```

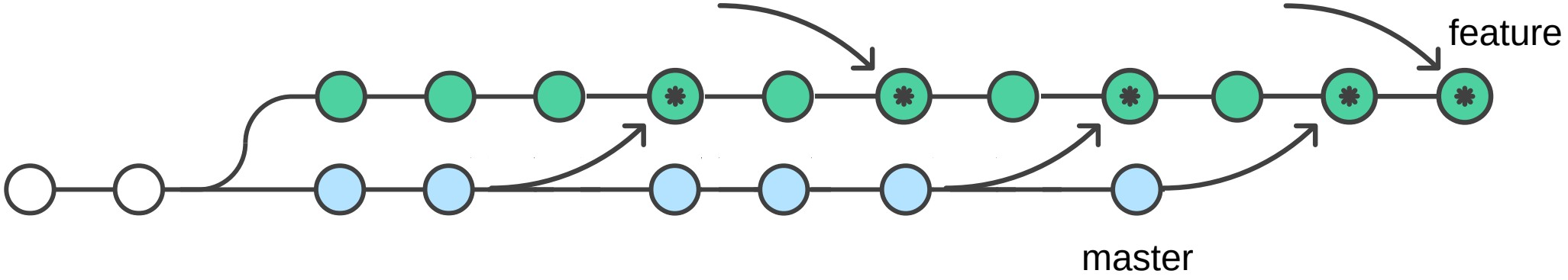
git merge

– If you need to incorporate upstream changes often,
your branch history can easily become quite ***unreadable***.



git merge

– If you need to incorporate upstream changes often,
your branch history can easily become quite ***unreadable***.



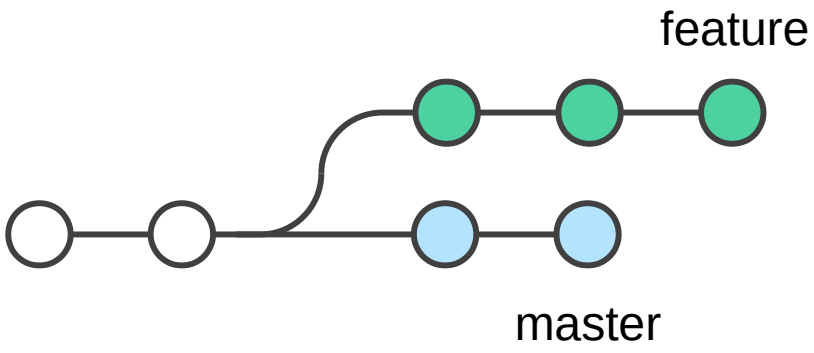
This makes hard for other developers
to follow the project history.

git merge


Merge branch '7475/ejabberd_wrap_pam_authentication_to_avoid_fai
Merge branch '7475/ejabberd_wrap_pam_authentication_to_avoid_fai
Merge branch 'feature/gui-Kalliope-308/alarm-clock' into 'master
Merge remote-tracking branch 'upstream/feature/gui-Kalliope-308/
Merge branch 'fix/port-4.7.17-to-master' into 'master'
Merge remote-tracking branch 'upstream/release/4.7.x' into fix/p
Merge branch 'reworking/KGUI-45/remove-link-to-kallioe-web-site-
gui: remove the link to KalliopePBX in login page
Merge branch 'fix/KGUI-47/fix-broken-SmsService' into 'master'
gui: fix wrong parameter
Merge branch 'feature/KGUI-46/fix-wrong-translation-istaza' into
gui: fix wrong translations
Merge branch 'feature/KGUI-16/remove-old-sendBusyUnavailableSMS-
gui: remove old sendBusyUnavailableSMS API
gui: add sendWiSms POST API
Merge branch 'feature/KGUI-26/add-delete-REST_API-for-active-ala
Merge branch 'feature/KGUI-25/add-delete-action-for-active-alarm
Merge branch 'feature/KGUI-43/modify-new-alarm-clock-instance-PO
Merge branch 'feature/KGUI-42/add-alarm-clock-list-GET-API' into
Merge branch 'feature/KGUI-21/new-terminated-alarm-clock-instanc
Merge branch 'feature/KGUI-40/alarm-clock-list-panel-must-show-o
Merge branch 'feature/KGUI-23/new-alarm-clock-instance-panel' in
Merge branch 'feature/KGUI-24/new-alarm-clock-instance-post-API'
Merge branch 'feature/KGUI-20/add-alarm-clock-instances-list-pan
gui: add DELETE API for alarm clock instance
gui: add delete action in alarm clock panel list
gui: modify new alarm clock instance REST API
gui: add REST API for alarm clock list
gui: add terminated alarm clock instance panel
gui: add parameters to filter function for terminated and not te
gui: add tests for new alarm clock instance panel
gui: add panel to start new alarm clock instance
gui: add trait to check alarm clock license
gui: add tests for new alarm clock instance REST API
gui: add REST API to start new alarm clock instance
gui: add translations
gui: add alarm clock instance list panel
gui: add alarm clock operation action
Merge branch 'feature/KGUI-35/modify-unique-contraint-for-call-c
Merge branch 'feature/KGUI-19/add-alarm-clock-edit-panel' into '
[3 notes]
[2 notes]
[1 note]
Merge branch 'feature/BaseFS-22/add-action-for-alarm-clock-configuration' into 'f
Merge branch 'feature/BaseFS-40/extend-campaign_instances-table-with-type-service
Merge branch 'feature/BaseFS-21/extend-call-campaign-service-with-type-service' i
Merge branch 'BaseFS-28/Add_missed_alarm_event_to_configurationdb' into 'feature/
BaseFS-28/Add_missed_alarm_event_to_configurationdb
Merge branch 'cherry-pick-24080868' into 'feature/fw_basefs-Kalliope-308/alarm-cl
Merge branch 'BaseFS-52/Add_event_type_to_configurationdb_for_kcallscheduler' int
Merge branch 'BaseFS-52/Add_event_type_to_configurationdb_for_kcallscheduler' int
BaseFS-52/Add_event_type_to_configurationdb_for_kcallscheduler
fw_basefs: add check on channel type
fw_basefs: add delete grant to call_campaign_status_write_role role
Merge branch 'bug/BaseFS-47/Add_cronjob_to_truncate_var_mail_files' into 'release
Merge branch 'fix/BaseFs-48/Reduce_timeout_for_repeat_request' into 'feature/fw_b
fix/BaseFs-48/Reduce_timeout_for_repeat_request
Feature/BaseFS-23/handle_campaign_type_alarmclock
bug/BaseFS-47/Add_cronjob_to_truncate_var_mail_files
fw_basefs: add check on SIP account prefix
Merge branch 'fix/BaseFS-45/fix-kcallscheduler-init-script' into 'feature/fw_base
* fixed the service name from amiproxy to callscheduler in init.d script
fw_basefs: add alarm clock operation action migration
fw_basefs: change unique index on call campaign entity
Merge branch 'revert-faedfde4' into 'feature/fw_basefs-CARAB-1'
Revert "Merge branch 'TEMP_add_kcallscheduler_binary_to_fw' into 'feature/fw_base
Merge branch 'feature/fw_basefs-CARAB-1' of https://gitlab.netresults.intranet:16
feat/BaseFS-44/Add_kcallscheduler_binary_to_the_firmware
Merge branch 'feature/basefs-11/kcallscheduler-under-safe' into 'feature/fw_basef
fw_basefs: add migration for the new action
fw_basefs: add type column in campaign_instances table
fw_basefs: add data initialization gor global settings
fw_basefs: add type column in call_campaign_settings_table a

git rebase

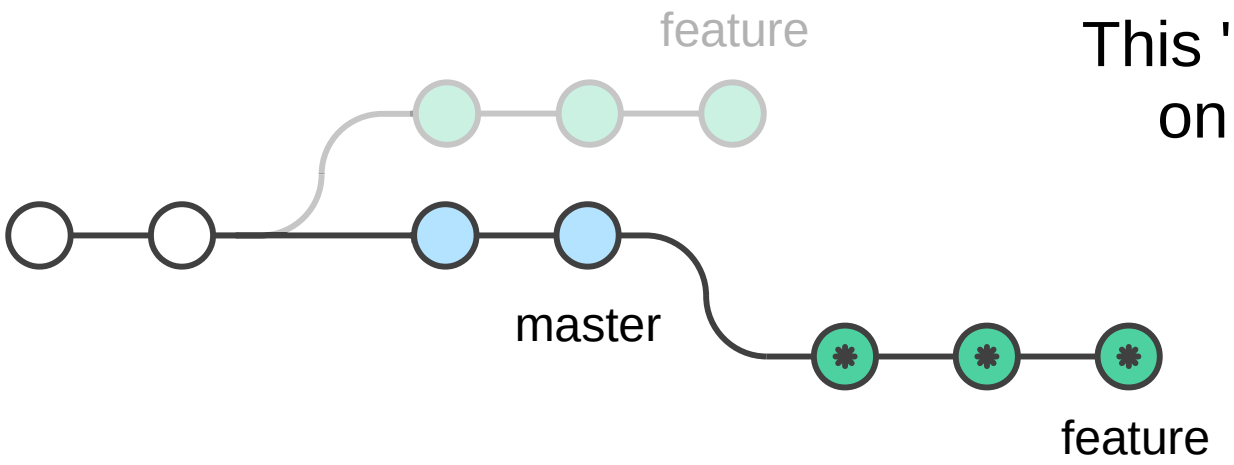
Rebase is another way to reintregrate changes from ***master*** branch into your ***feature*** branch.



git rebase

Rebase is another way to reintregrate changes from ***master*** branch into your ***feature*** branch.

= brand new commit



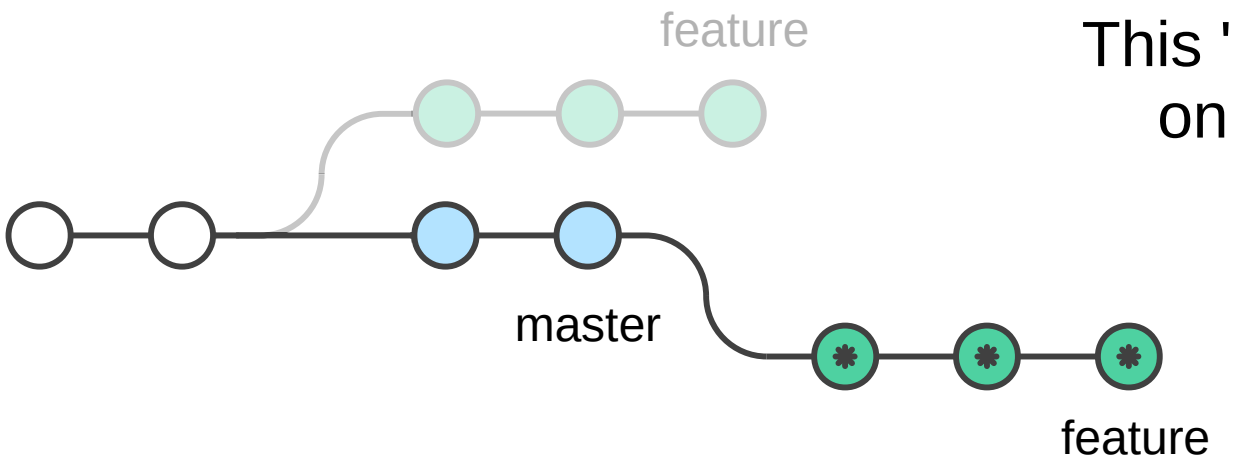
This "moves" ***feature*** branch to start on tip of ***master*** branch

```
git checkout feature
git rebase master
```

git rebase

Rebase is another way to reintregrate changes
from ***master*** branch into your ***feature*** branch.

= brand new commit



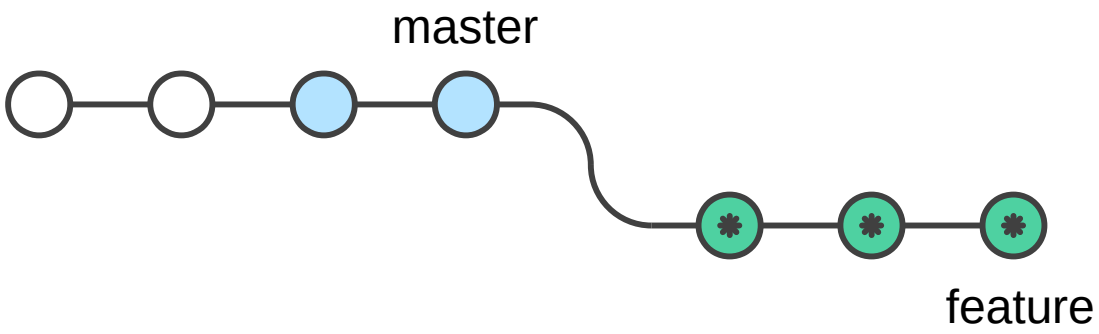
This "moves" ***feature*** branch to start
on tip of ***master*** branch

Rebase **re-writes** the history
by creating brand new commits.

```
git checkout feature
git rebase master
```

git rebase

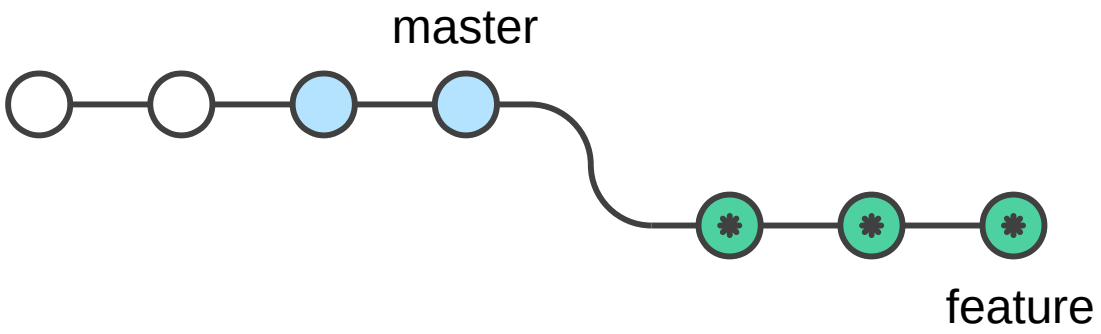
+ Rebase keeps the project history clean



git rebase

+ Rebase keeps the project history clean

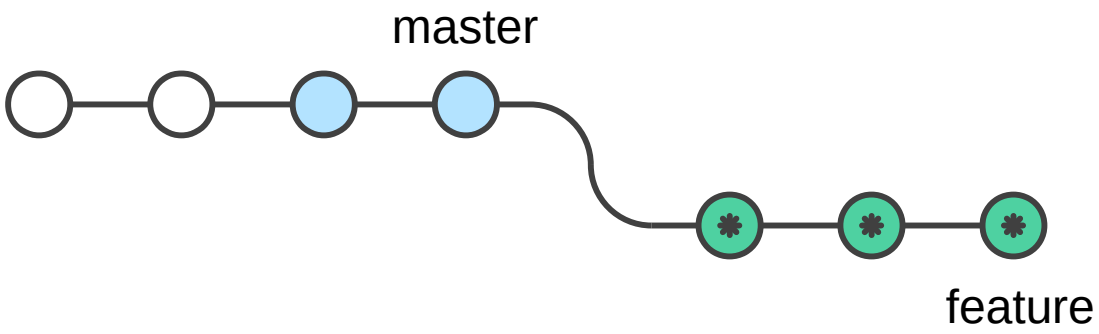
Resulting project history is **linear**,
with no forks (merge commit).



git rebase

+ Rebase keeps the project history clean

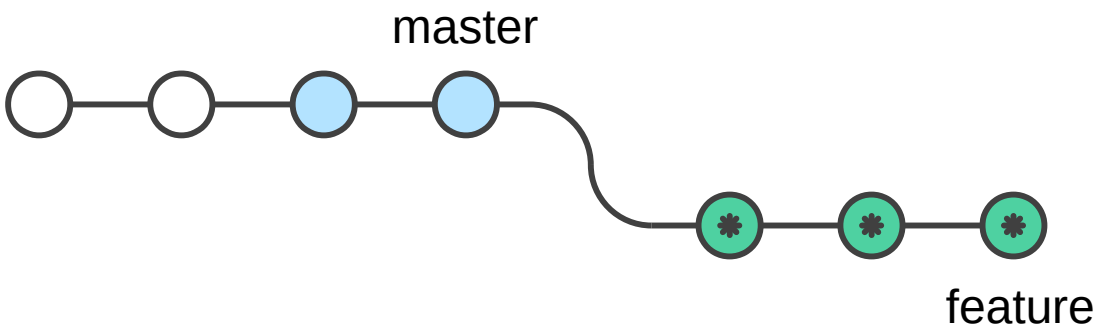
Resulting project history is **linear**,
with no forks (merge commit).



This makes easier to navigate history for everyone, even using tools like *git log.*

git rebase

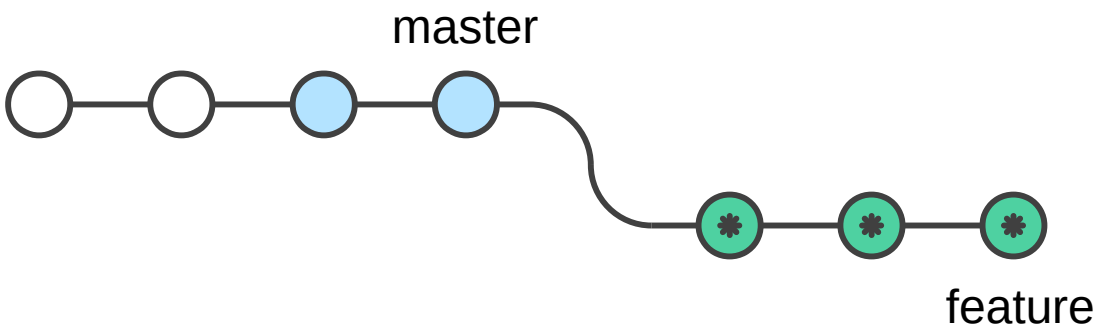
– Re-writing project history can be really catastrofic...



git rebase

– Re-writing project history can be really **catastrofic**...

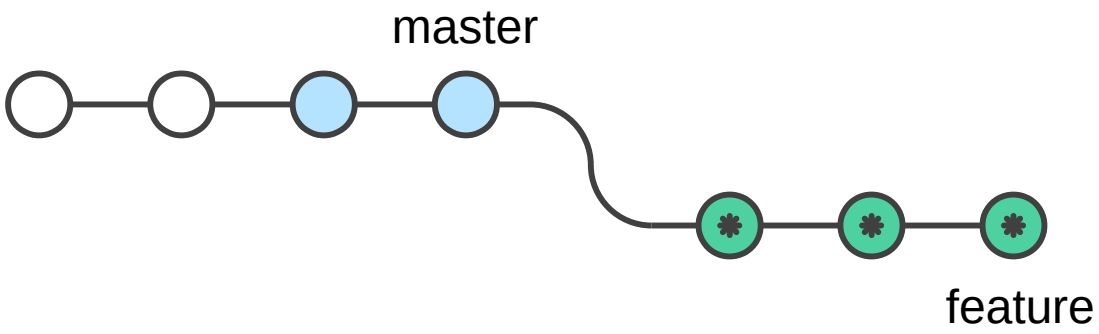
... **when** you're **collaborating** with other members **on the same branch**!



git rebase

– Re-writing project history can be really **catastrofic**...

... **when** you're **collaborating** with other members **on the same branch**!



Since rebase changes the project history, once a branch is rebased you may need to **force push** it to the remote repo.

WARNING: force push may cause lost of commits of the remote branch because it replaces the full branch history.

git r

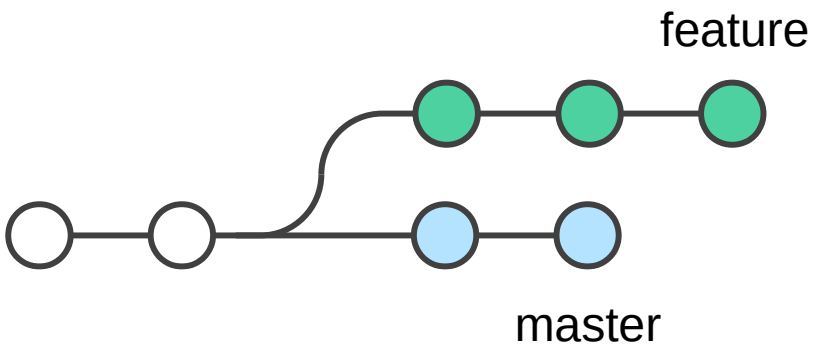
– Re-w

he project
s rebased
push it to

WARNIN
remote branch because it replaces the full branch history.

git rebase

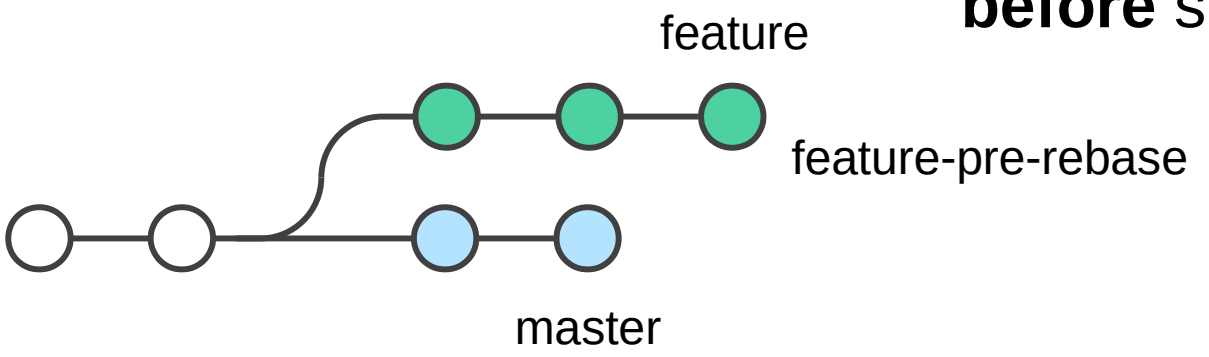
– Rebase cannot be undone...



git rebase

– Rebase cannot be undone...

... unless you create a new branch **before** starting rebase operation.

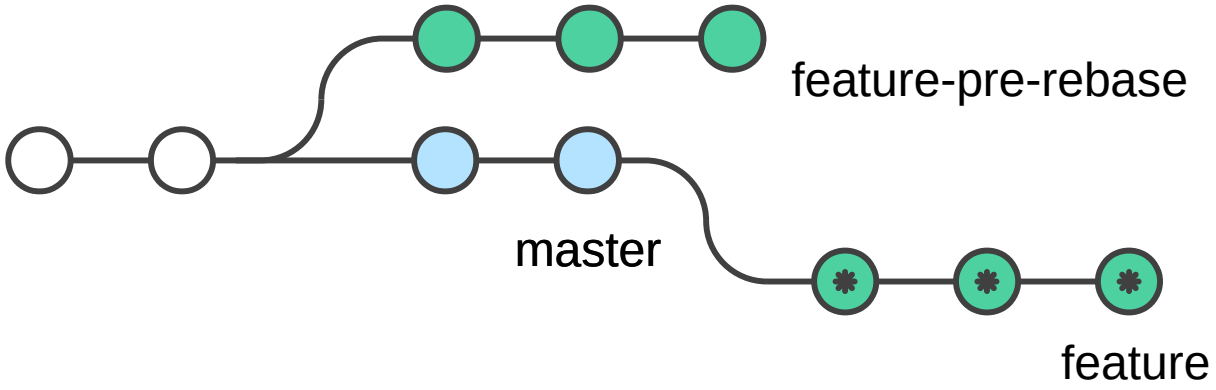



```
git checkout feature
git branch feature-pre-rebase
```

git rebase

– Rebase cannot be undone...

... unless you create a new branch **before** starting rebase operation.



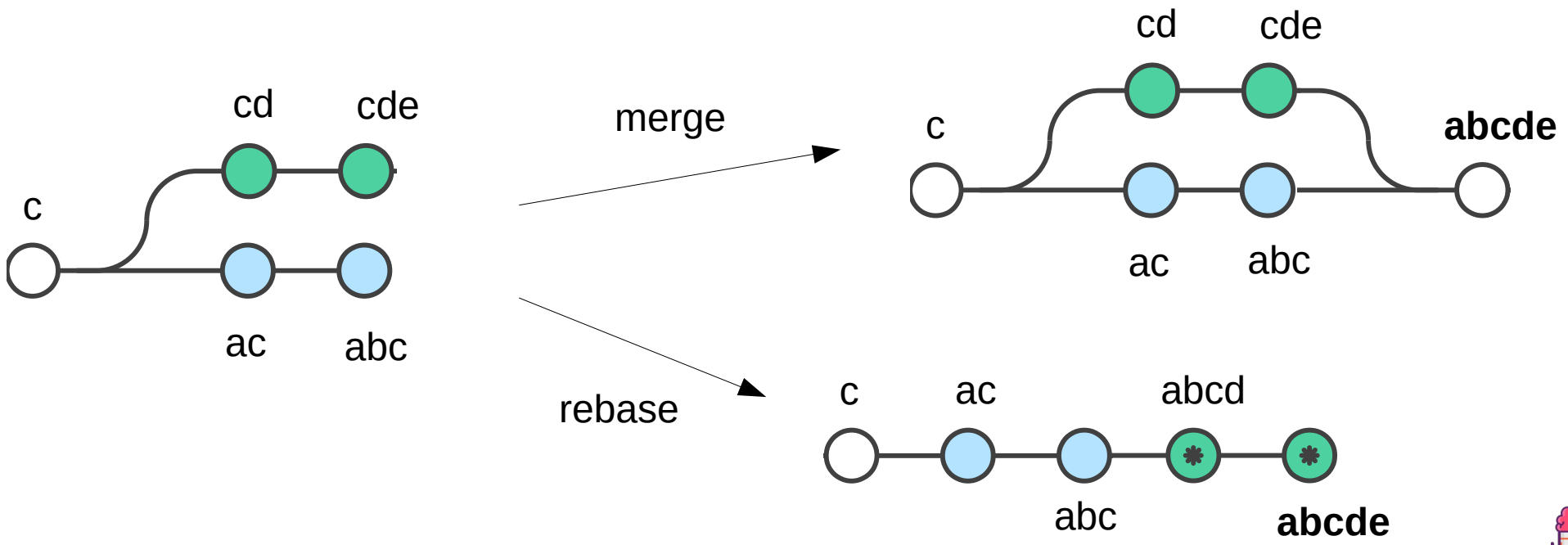
After rebase,
the ***feature-pre-rebase*** branch is still referencing the original branch.

```
git rebase master
```

git rebase vs. git merge

merge and **rebase** operations result in:

- the same files content
- with completely different history



git rebase vs. git merge

Rebase makes sense on **individual branches**.

Rebasing branches shared with other developers may cause lost of commits as well as inconsistent repositories.

> If you do so, you'll probably hear your colleagues in the other room invoking your name together with lots of saints.

Rebase allows you to completely rewrite commit history.

> Using **interactive rebase** you can easily **squash**, **drop**, **edit**, **add** and **reorder** commits on your branch.

git rebase vs. git merge

Merge is safe.

Use merge if you don’t know if someone else is working on the same branch.

Merge can be undone even if you forget to create a branch before it.

Merge preserves history.

> Using merge you can always see the history completely same as it happened.

git rebase vs. git merge (examples)

1. You are working on your local repo that is a clone of your remote repo (**origin**). Your **origin** is a fork of the **upstream** repo.

```
git clone <my_remote_repo>
git remote add upstream <upstream_repo>
```

Now, you want to start to work on a new feature.
Shared feature branch on **upstream** repo is not required for that feature.

```
git fetch upstream master
git checkout -b new_feature upstream/master
```

git rebase vs. git merge (examples)

After few commits you need to reintegrate changes from **upstream master**

```
git fetch upstream master
git rebase upstream master
```

After a couple of new commits, you push your work to your **origin**...

```
git push origin HEAD
```

...and then you create a new merge request.

git rebase vs. git merge (examples)

Unfortunately there are merge conflict.
To solve them you can rebase your branch.

```
git checkout new_feature          git checkout new_feature
git fetch upstream master   OR    git pull --rebase upstream master
git rebase upstream master
```

Solve conflicts, then push again...

```
git push origin HEAD
```

… and your push is rejected!

git rebase vs. git merge (examples)

As you changes the history, you must force push changes to your **origin**.

```
git push -f origin HEAD
```

Now merge request can be merged.

You can also merge from **upstream master** instead of rebase.

```
git checkout new_feature
git fetch upstream master
git merge upstream master
```

OR

```
git checkout new_feature
git pull upstream master
```

git rebase vs. git merge (examples)

2. Now, consider you need to work on a new feature having a shared feature branch on **upstream** repo.

```
git fetch upstream shared_feature
git checkout -b shared_feature upstream/shared_feature
```

After a couple of new commits, you push your work to your the shared branch on the **upstream** repo...

```
git push upstream HEAD
```

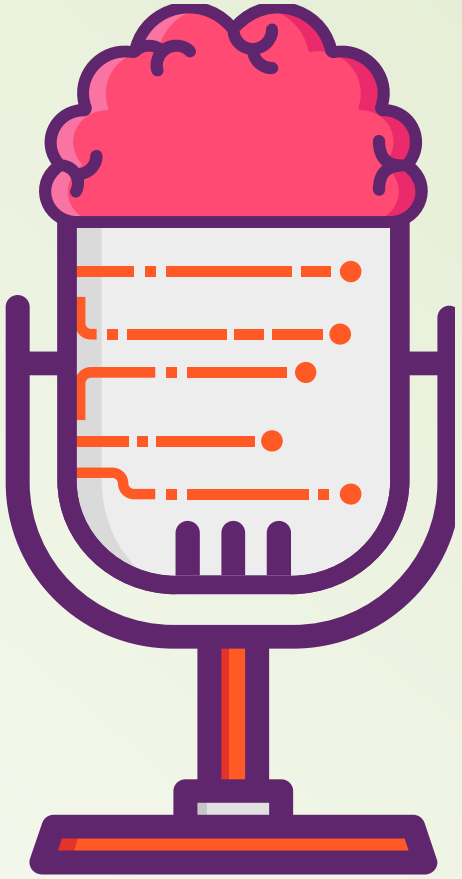
git rebase vs. git merge (examples)

Unfortunately your push is rejected!
Another member pushed changes that conflict with yours.

To solve the conflict you can rebase you local changes from **upstream** shared branch and push result to **upstream** repo.

```
git checkout shared_feature
git fetch upstream shared_feature
git rebase upstream shared_feature
git push upstream shared_feature
```

WARNING: never **force push** to shared branches!
If your push is rejected again, just rebase it and retry.



That’s all folks!

Questions?