Radboud University

Advance Use of Git

Benoît Viguier

DS-Lunch Talk,
Nijmegen, October 25th, 2019

Disclaimer

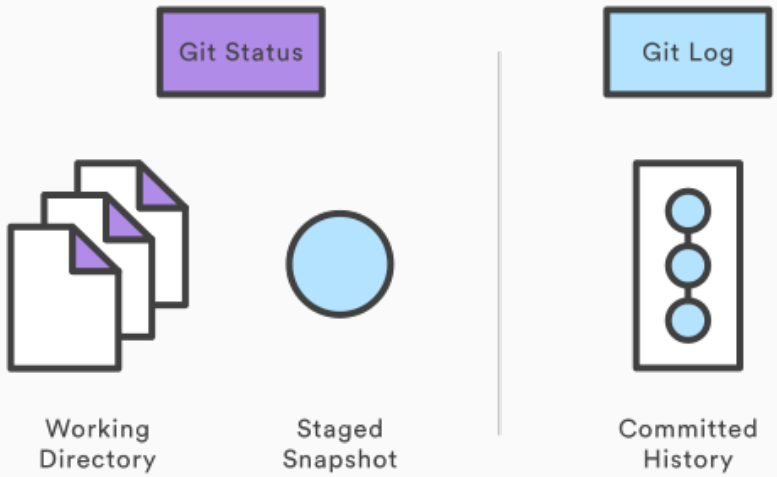
If you don't line command line interface,
this talk is not for you.

Minimal Git Commands

- `git clone git@gitlab.science.ru.nl:user/repo`
- `git status`
- `git add <directory/files>`
- `git commit`
- `git push`
- `git pull`

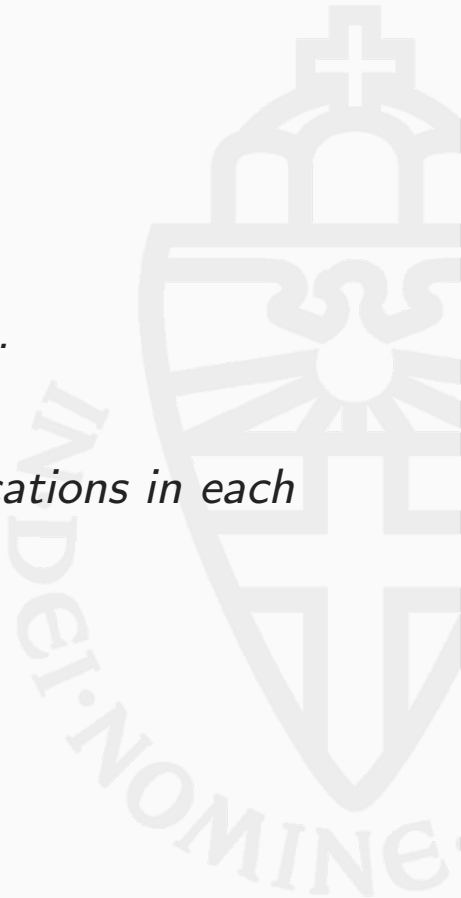
Git Status

- `git status`
- `git log`



Shortcuts

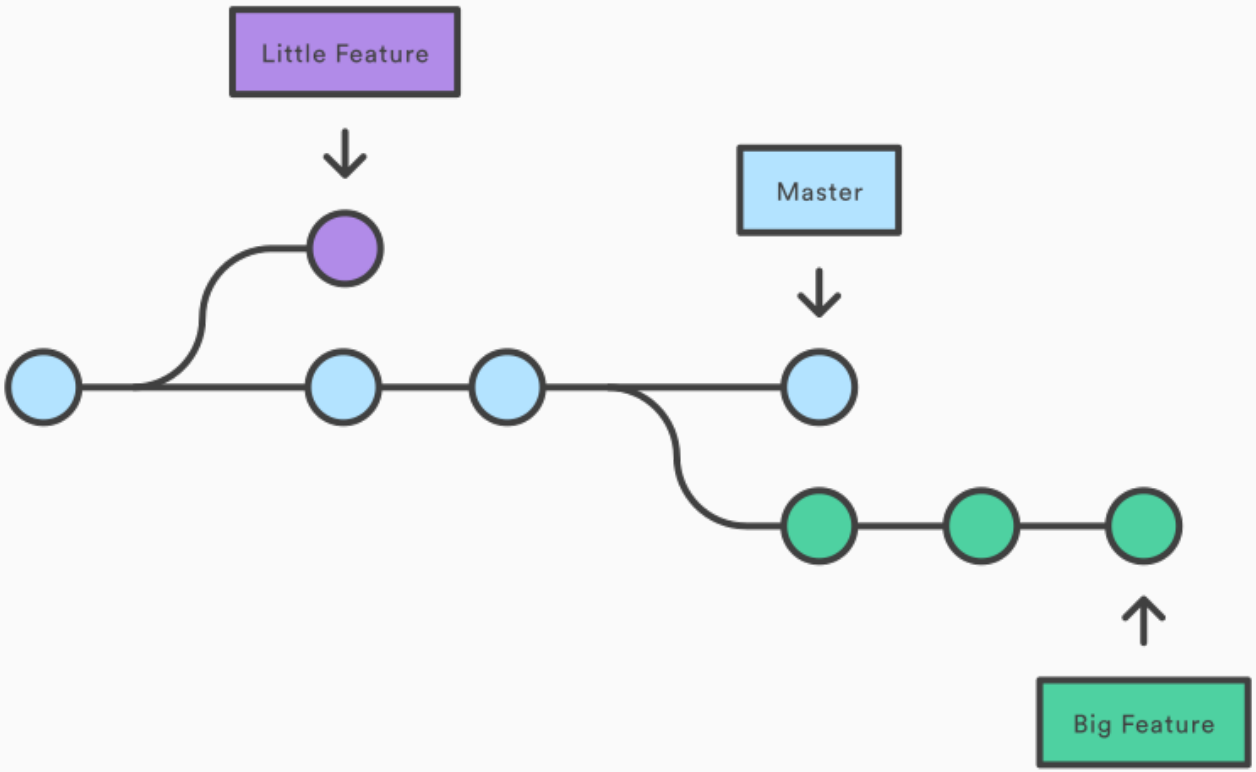
- `git add --all` (`-A`)
  Add all the updated/untracked files.
- `git add --update` (`-u`)
  Add all the updated files but do not add the untracked ones.
- `git add --patch` (`-p`)
  Similar to update. Interactively let you decide which modifications in each file you want to save.
- `git commit -m "commit message"`



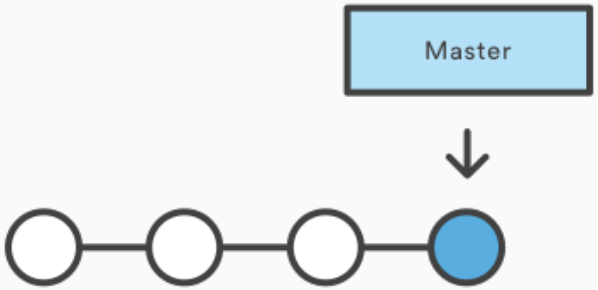
branches

Branches

- Develop features without breaking master.
  $\Longrightarrow$ the master branch always compiles! ✓
- Develop multiple features at the same time.

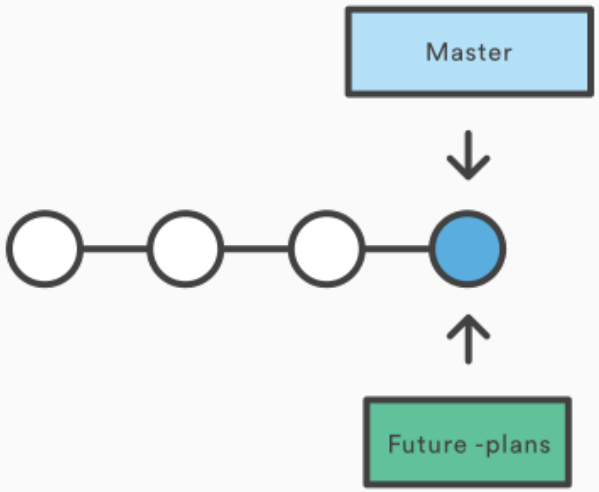


Branches



Branches

- Create a branch `git branch Future-plans`
- Switch to that branch `git checkout Future-plans`

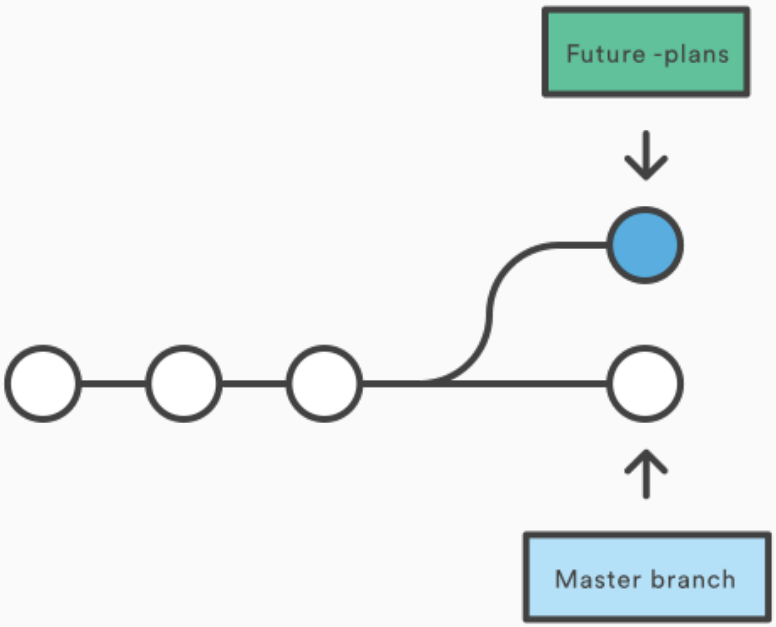


These two can be done in one step: `git checkout -b Future-plans`

Branches

Work (modify, commits...) on the `Future-plans`.

In the mean time, the `Master` branch continue forward (other commits...)

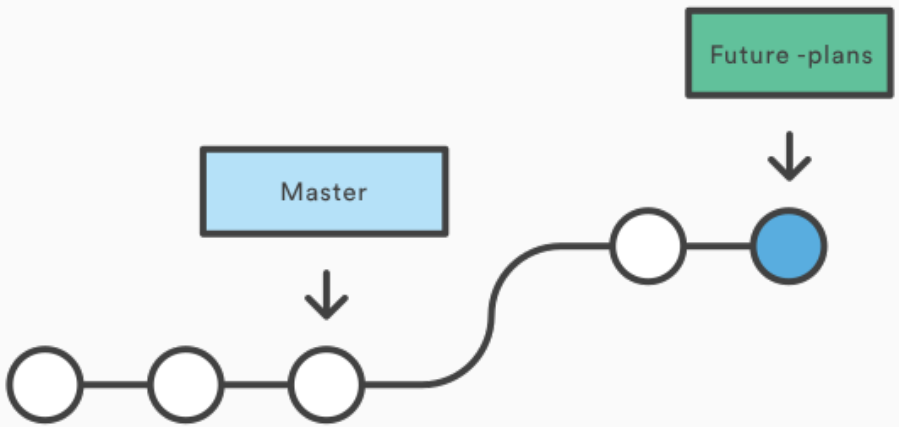


Branches

If nothing was done on `Master` while you were working on `Future-plans` you can directly merge. This is called *fast-forward*.

(1) Switch to the master branch: `git checkout master`

(2) Merge: `git merge Future-plans`

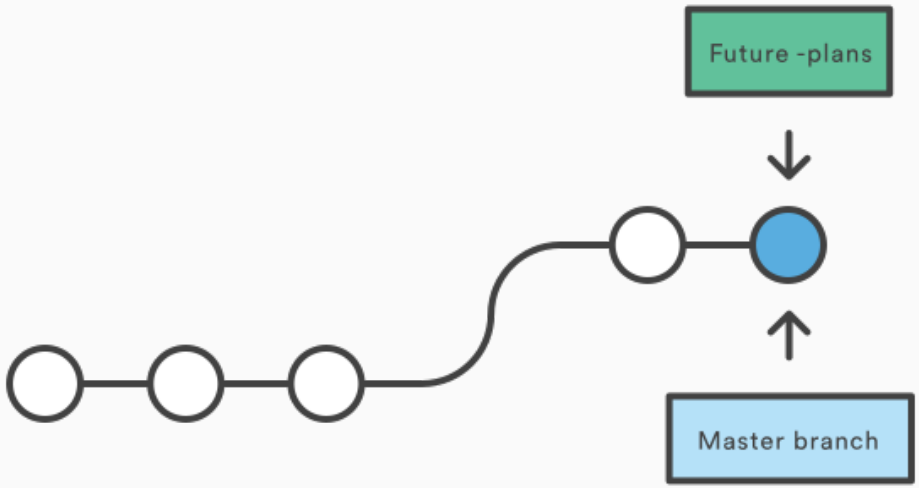


Branches

If nothing was done on `Master` while you were working on `Future-plans` you can directly merge. This is called *fast-forward*.

(1) Switch to the master branch: `git checkout master`

(2) Merge: `git merge Future-plans`



Branches

To push/update a branch on the online repository:

```
git push origin Future-plans
```

To delete (`-d`) a branch on the online repository:

```
git push -d origin Future-plans
```

To delete (`-D`) locally a branch:

```
git branch -D <branch-name>
```

(Not possible while you are on that branch.)

Merging

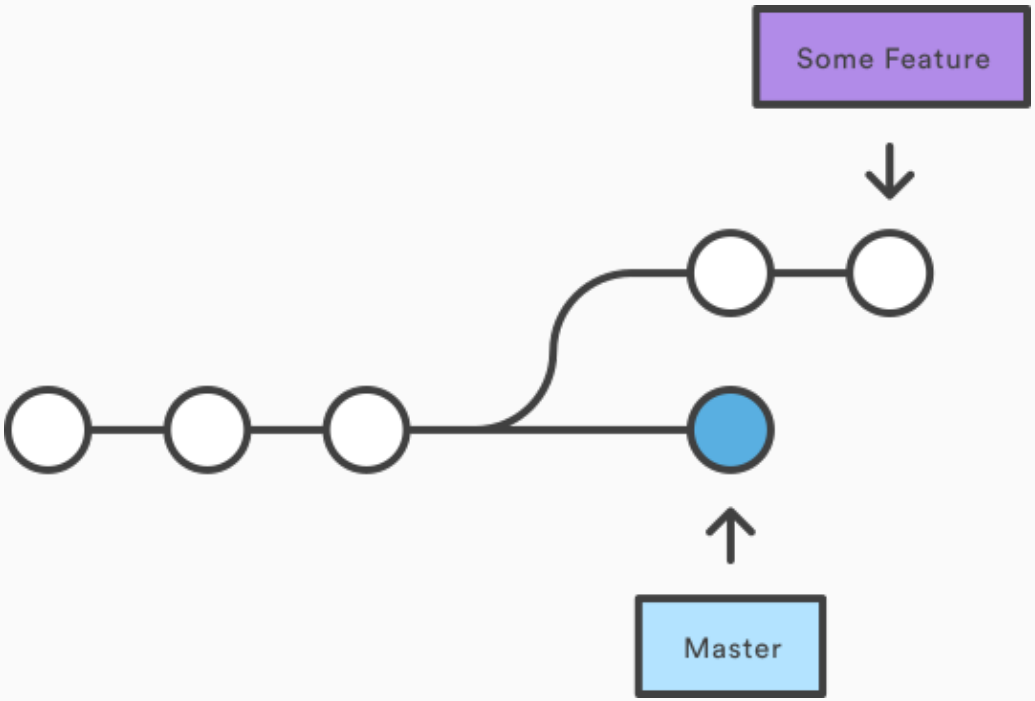
Merging

`Master` has changed while you were working on `Future-plans` the merge process is slightly different.

(1) Switch to the master branch: `git checkout master`

(2) Merge: `git merge Future-plans`

(3) Edit the commit message in your editor, save and close.



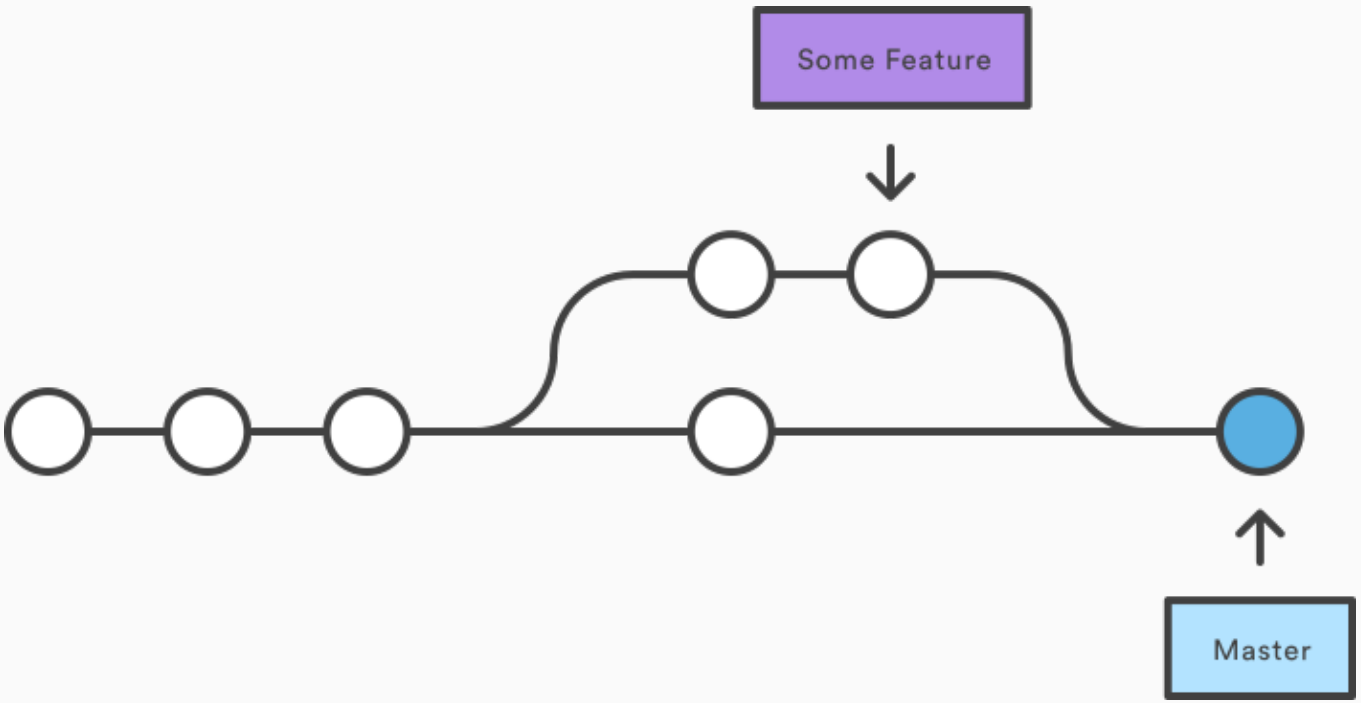
Merging

`Master` has changed while you were working on `Future-plans` the merge process is slightly different.

(1) Switch to the master branch: `git checkout master`

(2) Merge: `git merge Future-plans`

(3) Edit the commit message in your editor, save and close.

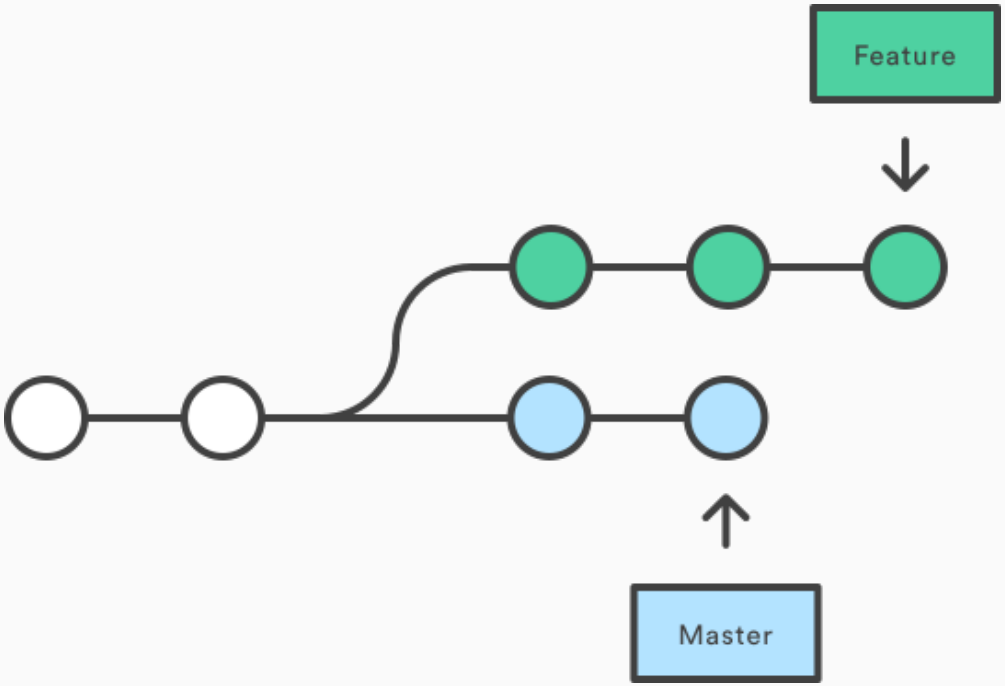


Rebasing

Merging & Rebasing

`Master` has changed while you were working on `Feature`.

You want to make sure your modification do not break `Master`.

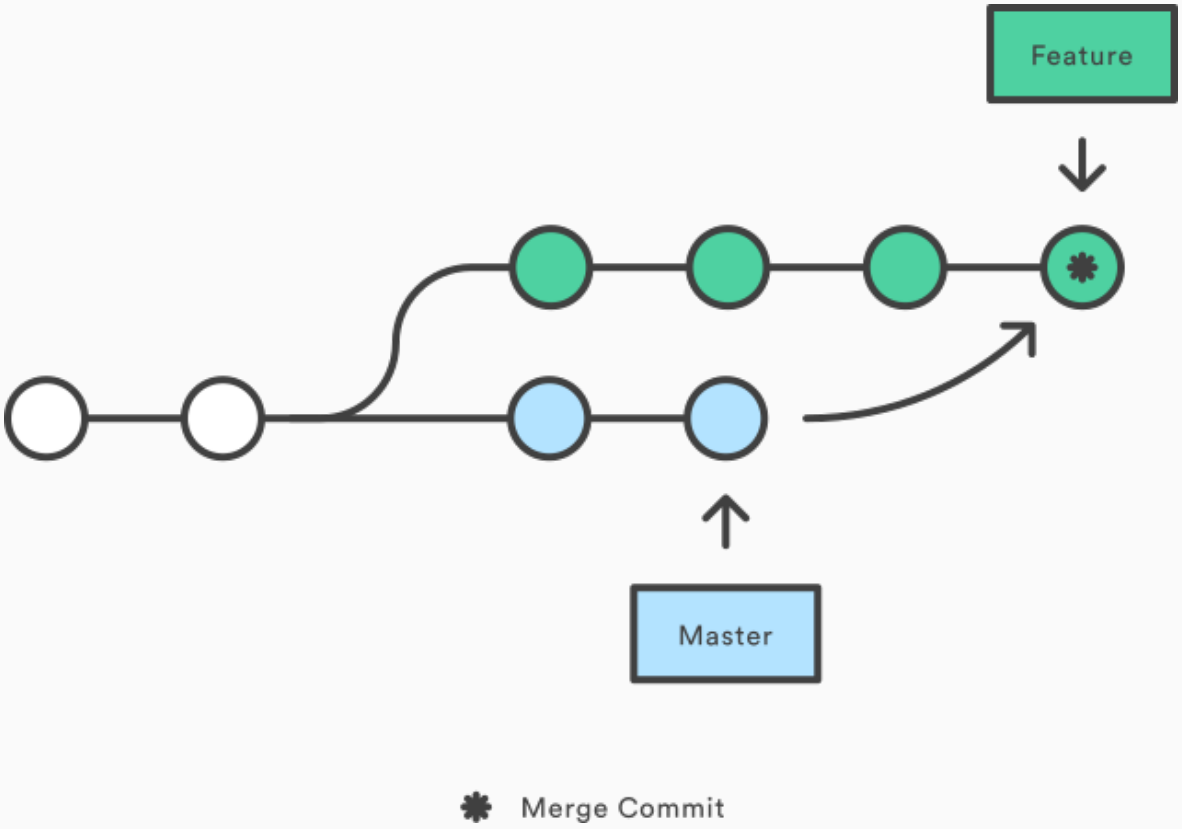


Merging

Solution 1:

(1) Switch to the Feature branch: `git checkout Feature`

(2) Merge: `git merge Master`

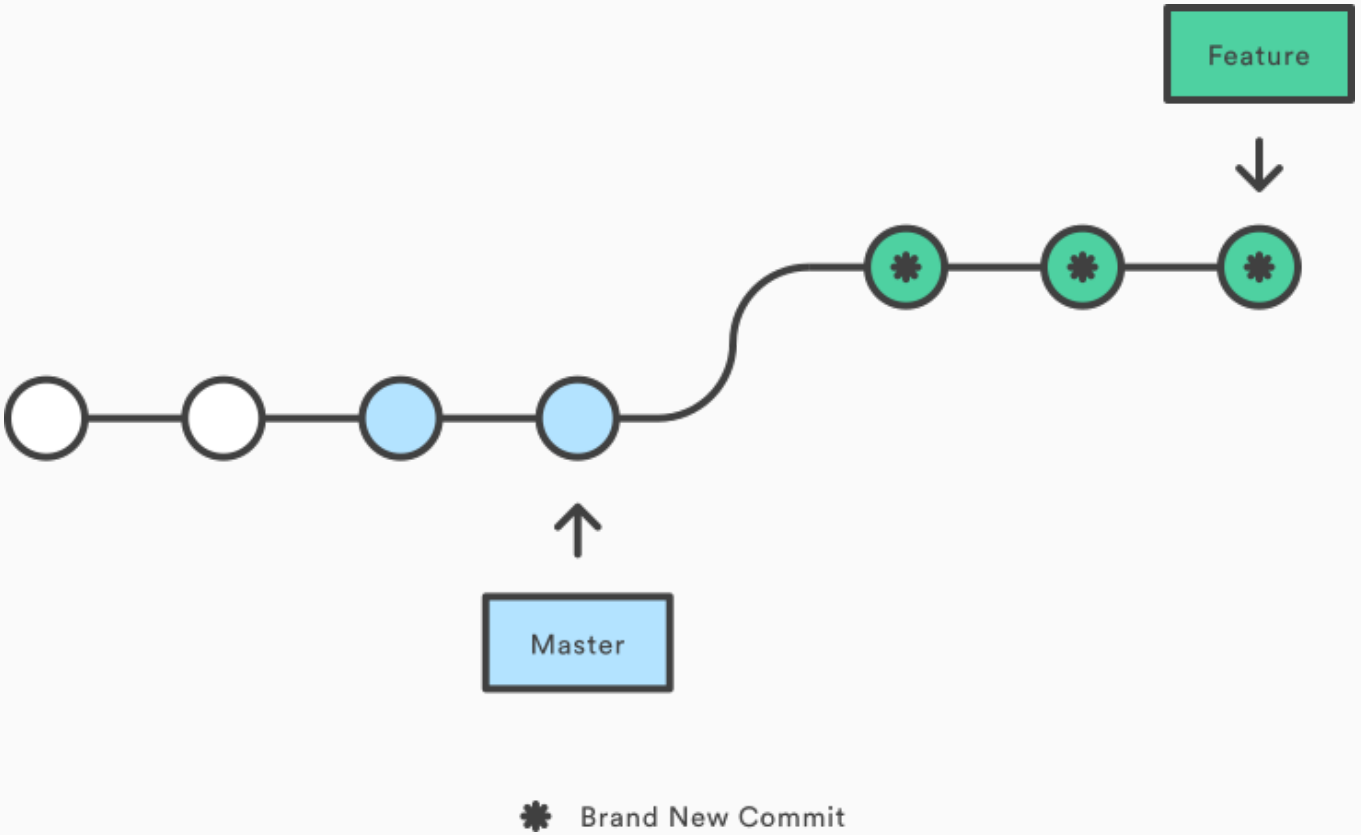


Rebasing

Solution 2:

(1) Switch to the Feature branch: `git checkout Feature`

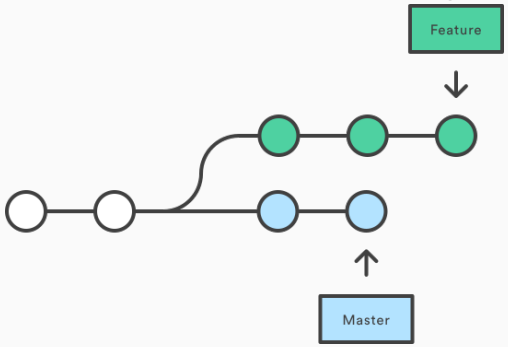
(2) Merge: `git rebase Master`



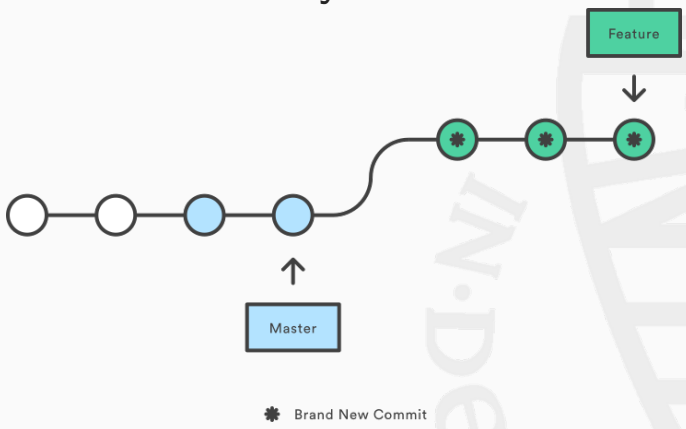
Force Pushing

Once you have rebased you have a conflict between your local tree and the remote tree.

What the remote repository knows:



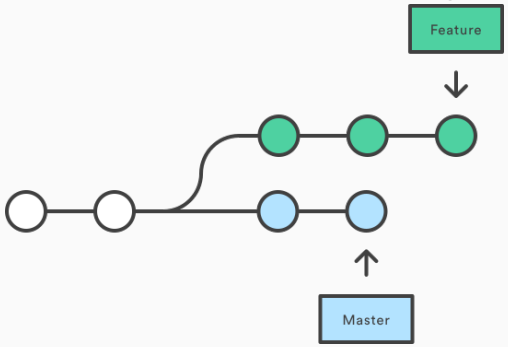
What you have:



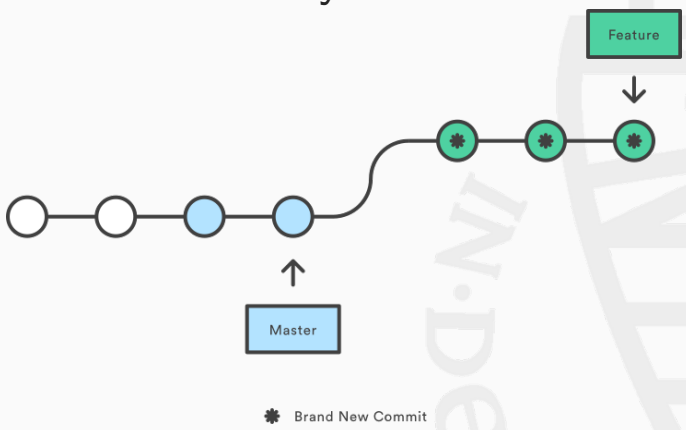
Force Pushing

Once you have rebased you have a conflict between your local tree and the remote tree.

What the remote repository knows:



What you have:



The solution is a force push:

```
git push --force origin Feature
```

Force Pushing

Force pushing is very dangerous and will break everything if not used correctly.

- **NEVER force push on Master.**
- **ALWAYS specify the repo and the branch.**

Conflicts

Conflicts

Conflicts can happen when you do a pull, merge or rebase.

```
git merge new_branch_to_merge_later
```

```
Auto-merging merge.txt
CONFLICT (content): Merge conflict in merge.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Conflicts

```
git status
```

```
On branch master
You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)

both modified:   merge.txt
```

Conflicts

```
cat merge.txt
```

```
...
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> new_branch_to_merge_later
...
```

Conflicts

Resolution steps:

(1) Edit the file: select the part you like, erase the alternative, save.

(2) `git add merge.txt`

(3) `git commit -m "Merged and resolved conflict"`

In the case of a `rebase`, instead of writting a commit message, just do `git rebase --continue`.

In both case, if you are not sure, you can use the `--abort` option.

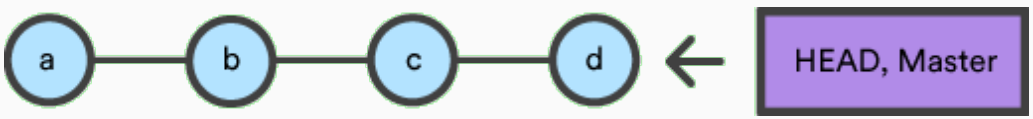
HEAD, Checking out, Reverting & Resetting

HEAD & checkout

The pointer of the current location in Git is called the **HEAD**. It can be used as a reference point.

E.g. if you want to go back to a previous commit, you can do either:

- `git checkout HEAD~2`
- `git checkout b` where `b` is a commit id

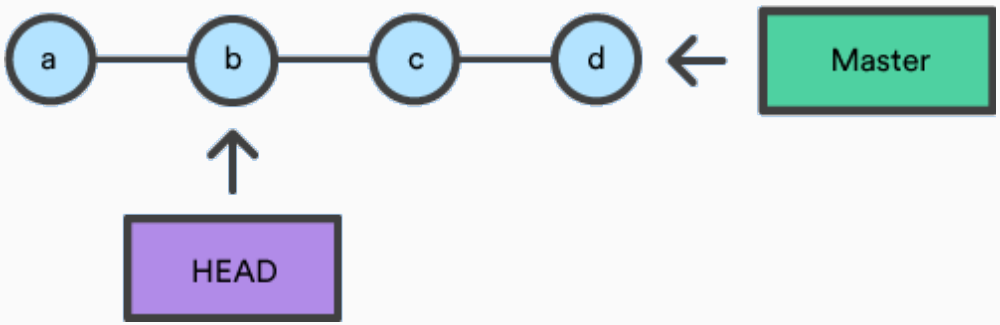


HEAD & checkout

The pointer of the current location in Git is called the **HEAD**. It can be used as a reference point.

E.g. if you want to go back to a previous commit, you can do either:

- `git checkout HEAD~2`
- `git checkout b` where `b` is a commit id



From there you can start working on a new branch: `git checkout -b Foo`

Changing the commit message

You can rewrite the message of the last commit with:

```
git commit --amend
```

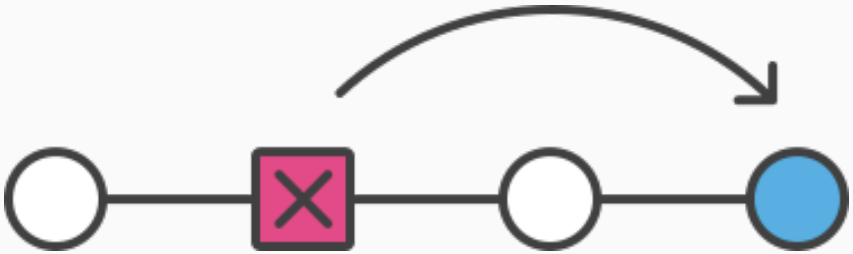
Changing the commit message

You can rewrite the message of the last commit with:

```
git commit --amend
```

Reverting

If you want to undo the last commit: `git revert HEAD`
This will **create** a new commit which reverts the last changes.

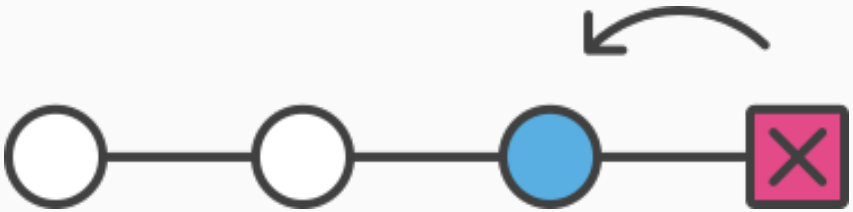


If you want to undo the change of an older commit you can also do:
`git revert commit_id` e.g. `git revert a1e8bf5` or
`git revert HEAD~1`

Resetting

`git reset --... commit_id` comes with 2 main options:

- `--soft` : keep the files as is but reset the pointer to `commit_id` .
- `--hard` : reset the files to the pointer `commit_id` .



Reset tricks

- `git reset --hard HEAD` : remove all the change made from the HEAD
- `git reset --soft HEAD~4` : go back and forget 4 commits but leave the files as is. Usefull if you want to squash your history.
- `git reset --hard commit_id` : set the repository as it was in `commit_id`

Stage, Diff, Stash, Clean

Staged files and diff

Files have 4 states:

- Commited
- Staged
- Unstaged
- Untracked

Staged files are contains changes that are reccorded by Git but not commited yet.

`git diff` will show the diff between staged/commited and unstaged files.

Stash

`git stash` comes with 4 main option:

- `push` : (optional) save your local modifications and revert to HEAD.
- `pop` : apply the modifications it on top of the current working tree state.
- `list`
- `clear`

During a `git stash ; git stash pop` , the all moficiations are unstaged.

Clean

You modified a lot of files, you have a lot of untracked files, your repository is dirty? Don't worry, `git clean` is here for you!

- `git clean -nd` : list the files to be removed
- `git clean -fd` : remove recursively (`-d`) untracked files
- `git clean -fxd` : remove recursively untracked and ignored (`-x`) files

By default `git clean` will do nothing, it requires either:

- `-n` for a dry-run.
- `-f` for force

Workflow: All on Master (classic academia)

All on Master

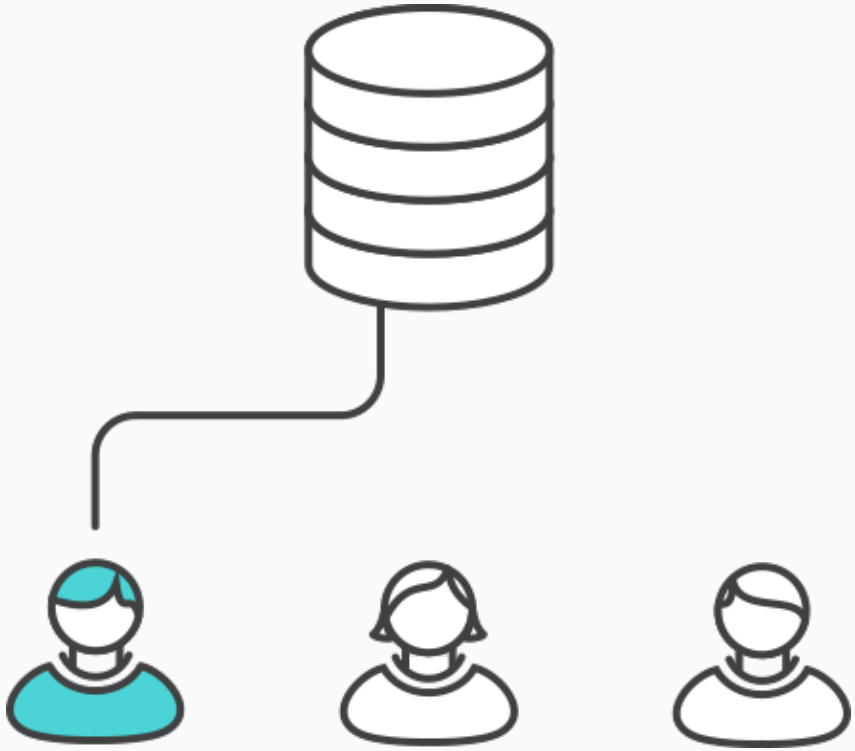


John works on his feature

All on Master

Mary works on her feature

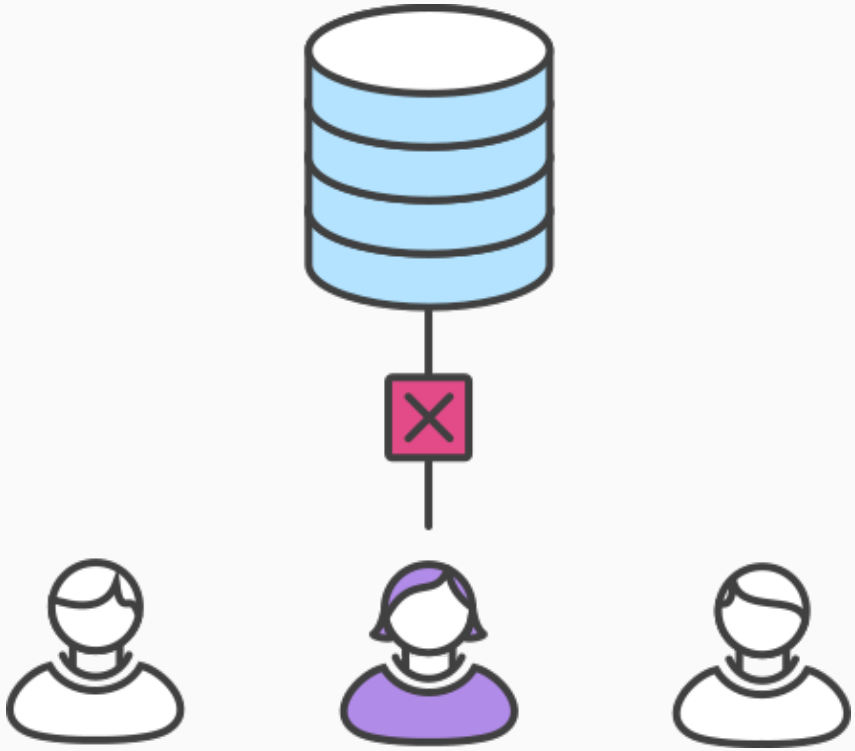
All on Master



John publishes his feature

```
git push origin master
```

All on Master

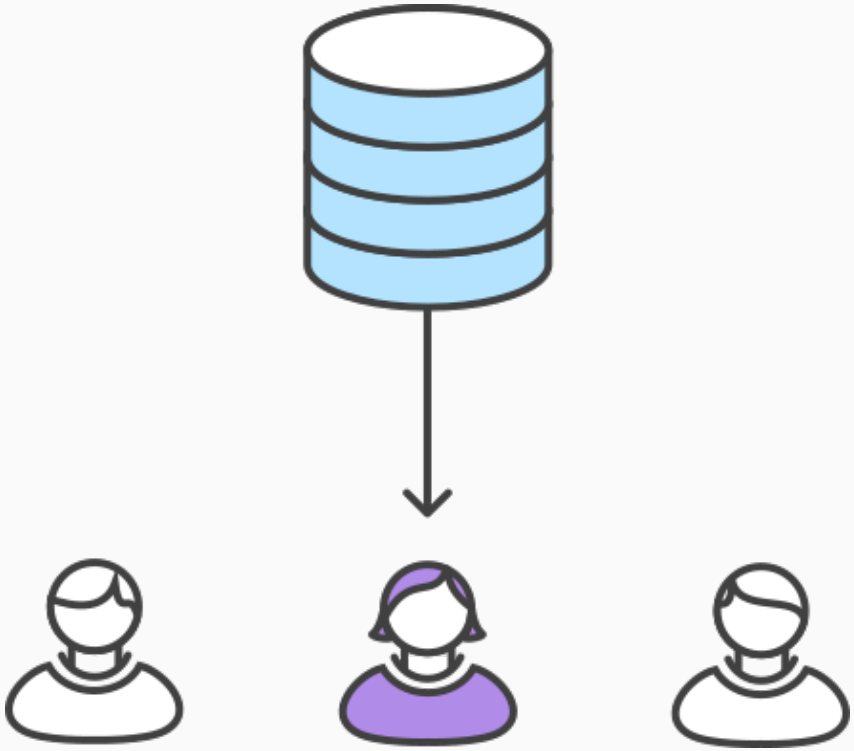


Mary tries to publish her feature

```
git push origin master
```

```
error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')
hint: before pushing again.
```

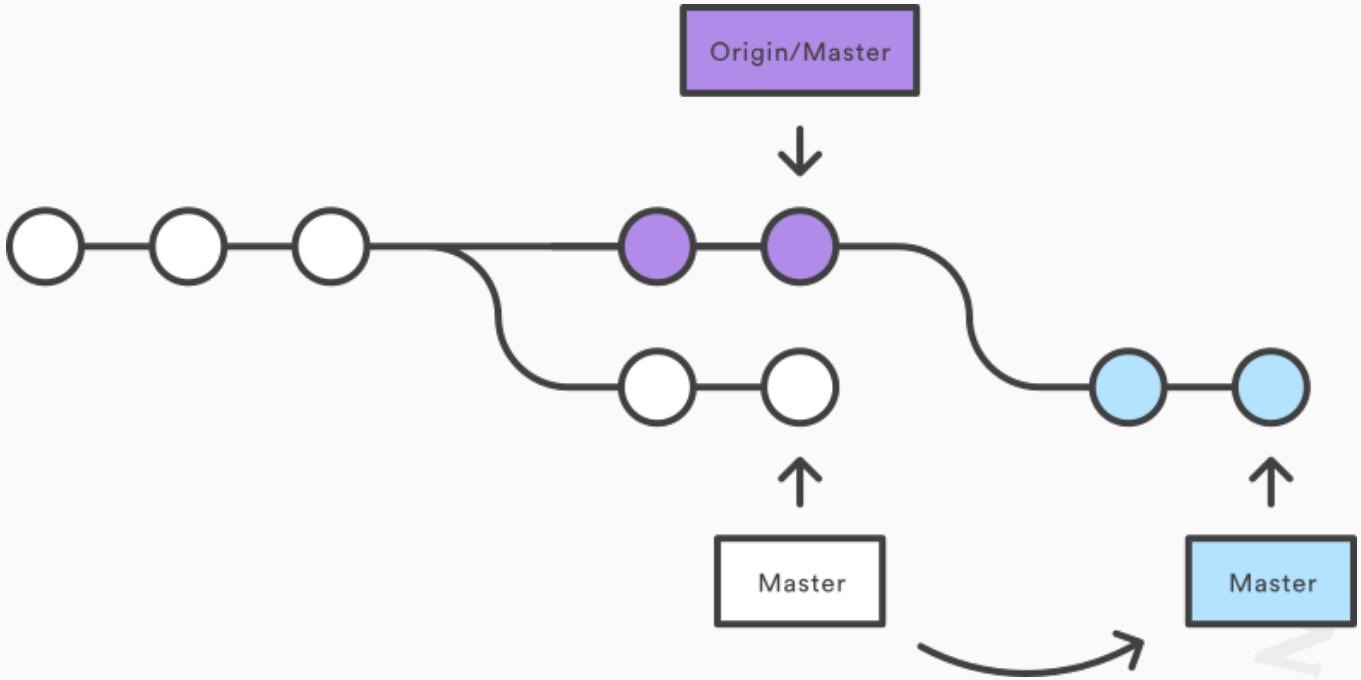
All on Master



Mary rebases on top of John's commit(s)

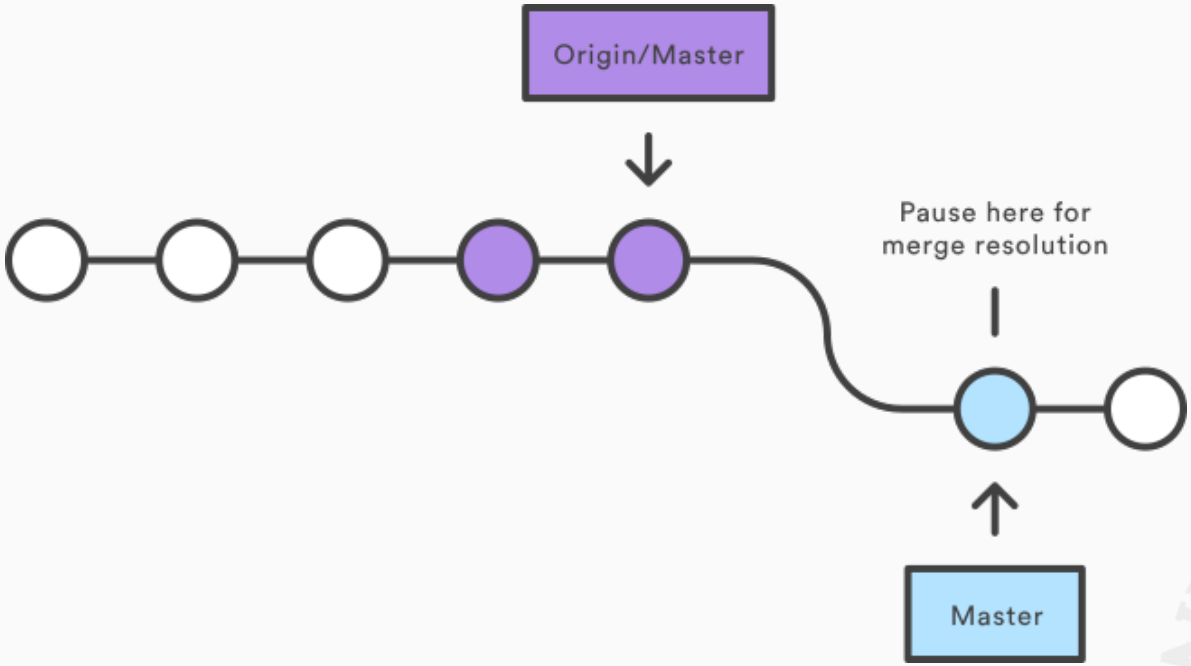
```
git pull --rebase origin master
```

All on Master



Expected new tree (Mary POV).

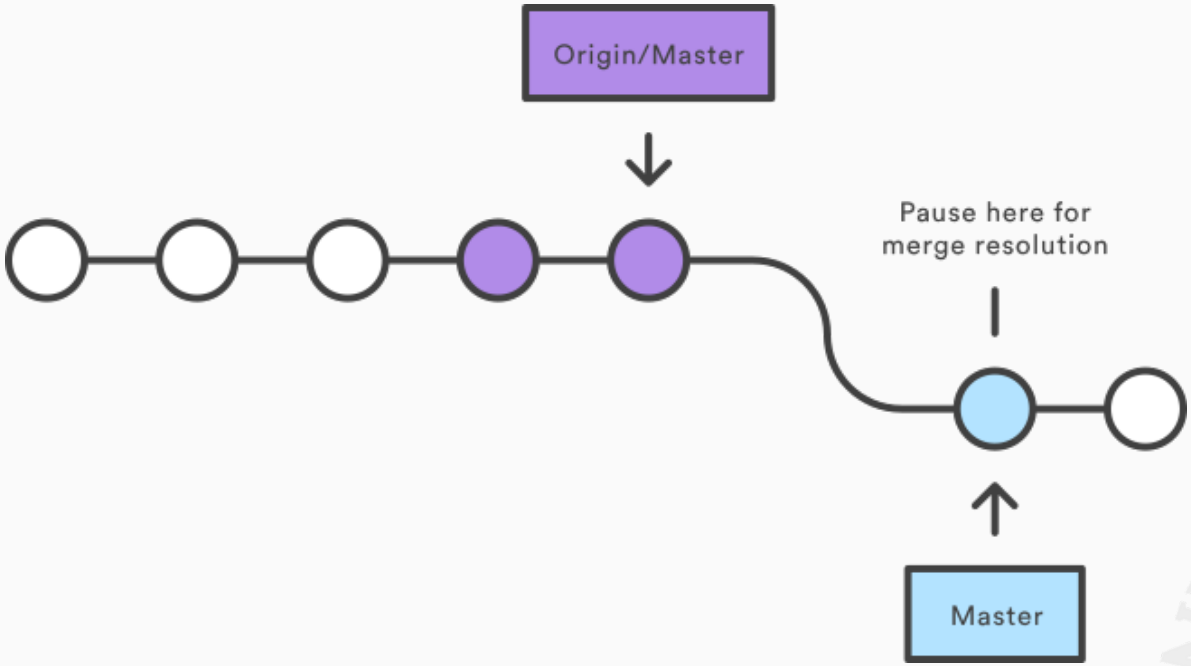
All on Master



But there is a conflict...

```
CONFLICT (content): Merge conflict in <some-file>
```

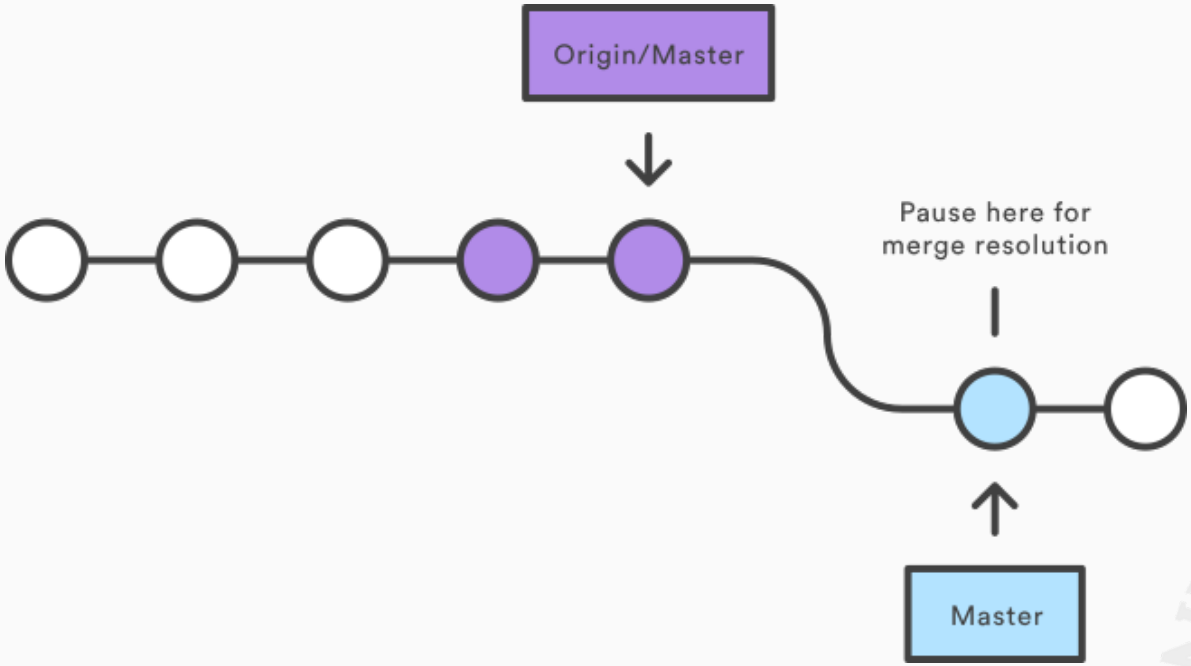
All on Master



`git status`

```
# Unmerged paths:
# (use "git reset HEAD <some-file>..." to unstage)
# (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
# both modified: <some-file>
```

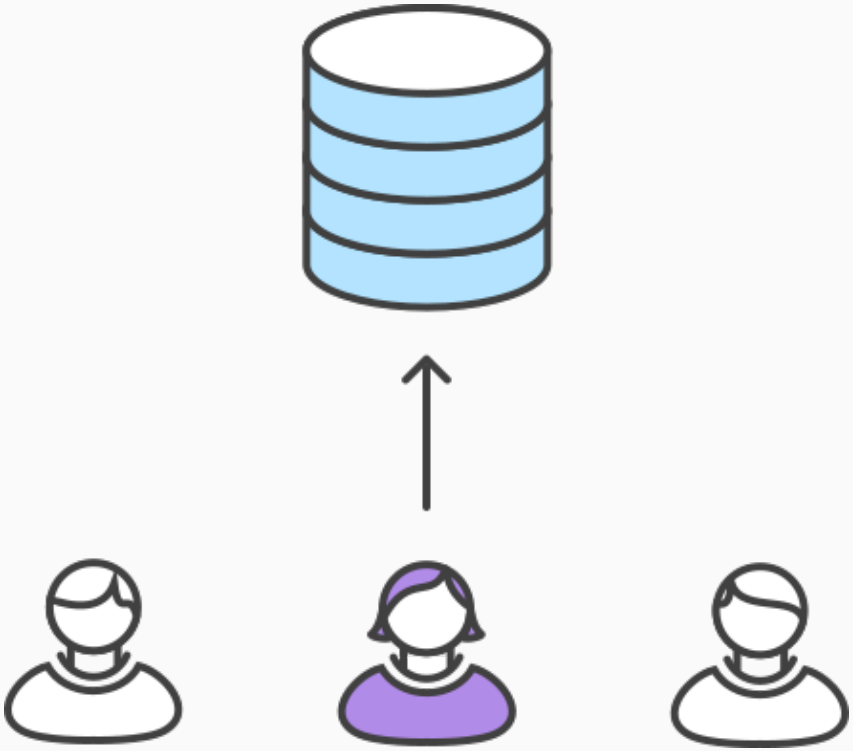
All on Master



Mary edits <some-file>

```
git add <some-file>
git rebase --continue
```

All on Master

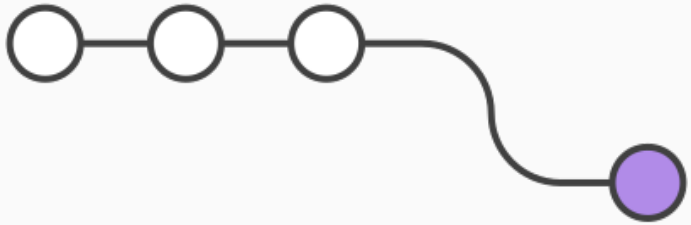


Mary successfully publishes her feature

```
git push origin master
```

Workflow: Branch Workflow

Using Branches and Pull Requests



Mary begins a new feature

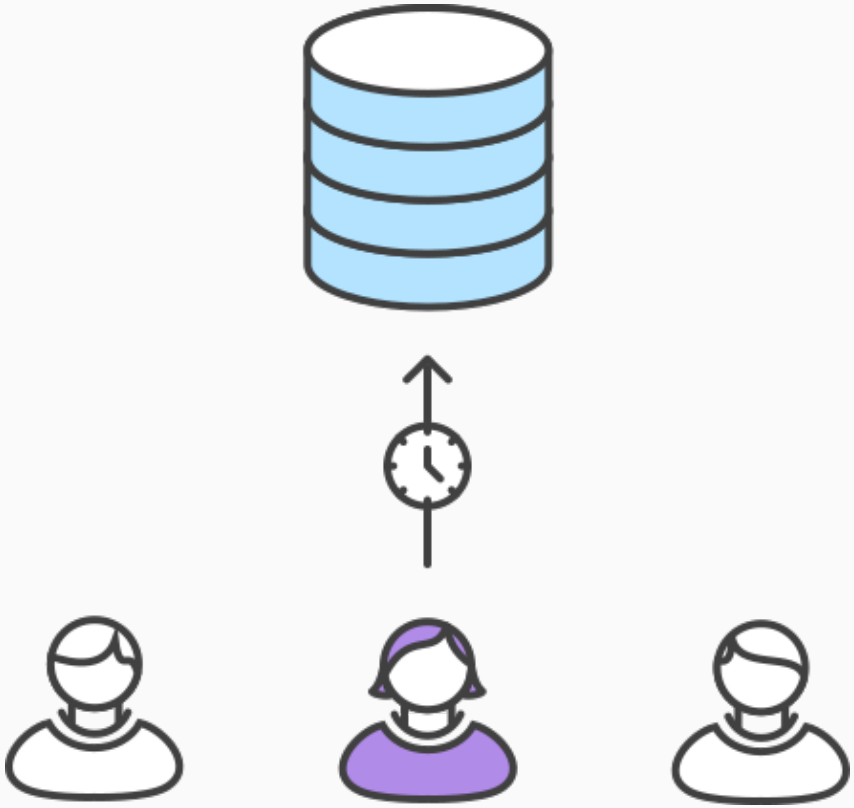
```
git checkout -b marys-feature master
```

```
git status
```

```
git add <some-file>
```

```
git commit
```

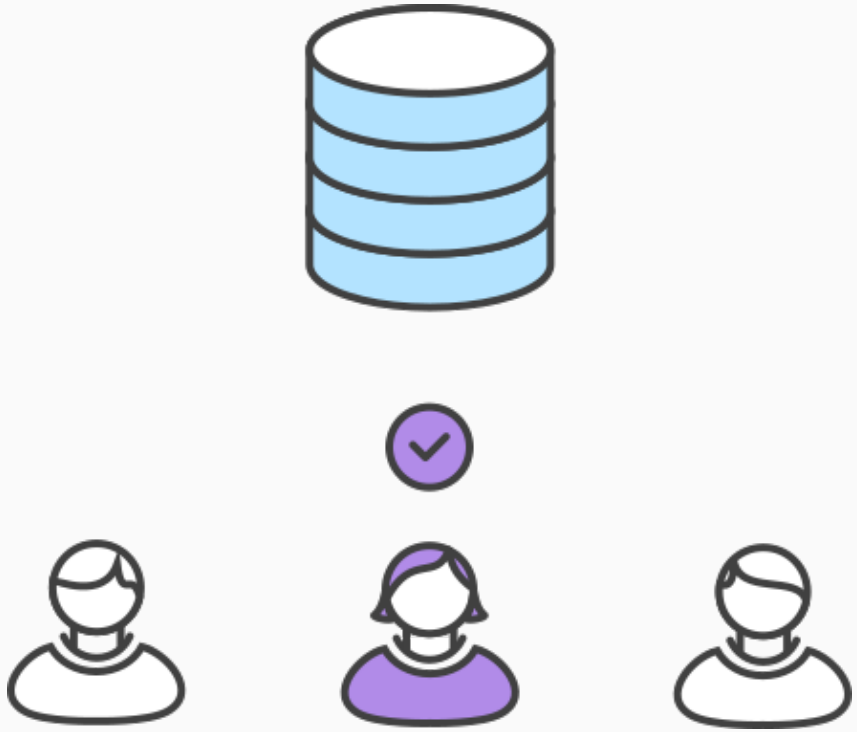
Using Branches and Pull Requests



Mary goes to lunch

```
git push -u origin marys-feature
```

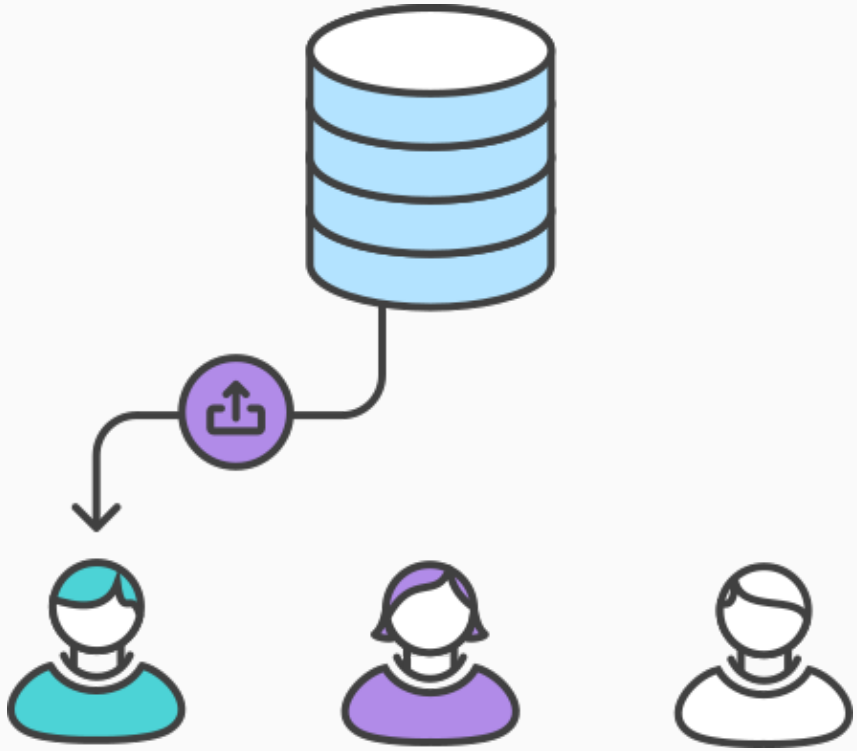
Using Branches and Pull Requests



Mary finishes her feature

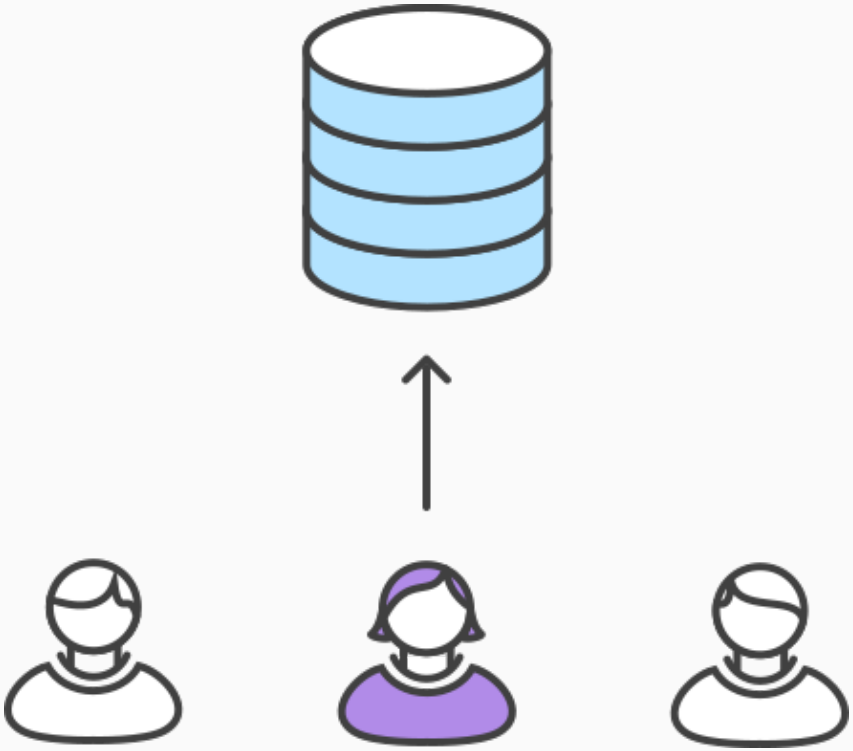
```
git push
```

Using Branches and Pull Requests



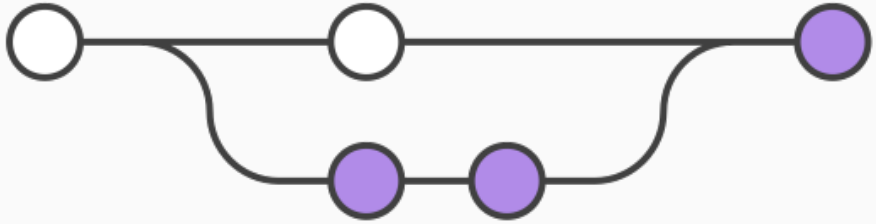
Bill receives the Pull Request, review and [ask for some change/comment/approve]

Using Branches and Pull Requests



Mary makes the changes

Using Branches and Pull Requests



Mary publishes her feature

```
git checkout master
```

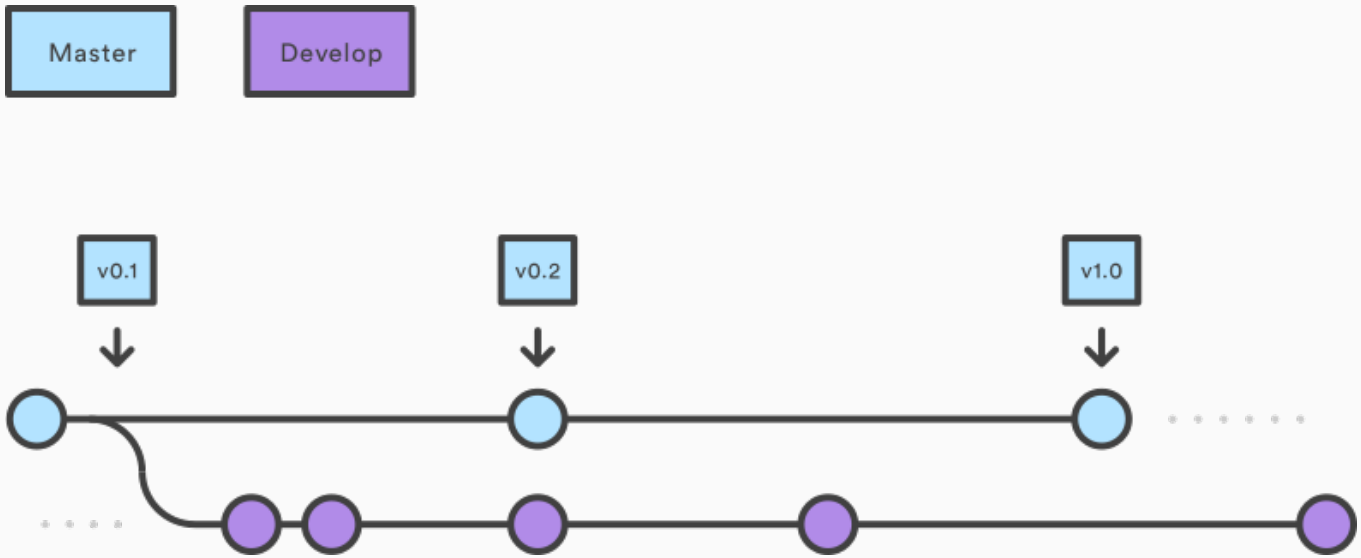
```
git pull
```

```
git merge marys-feature
```

```
git push
```

Gitflow Workflow

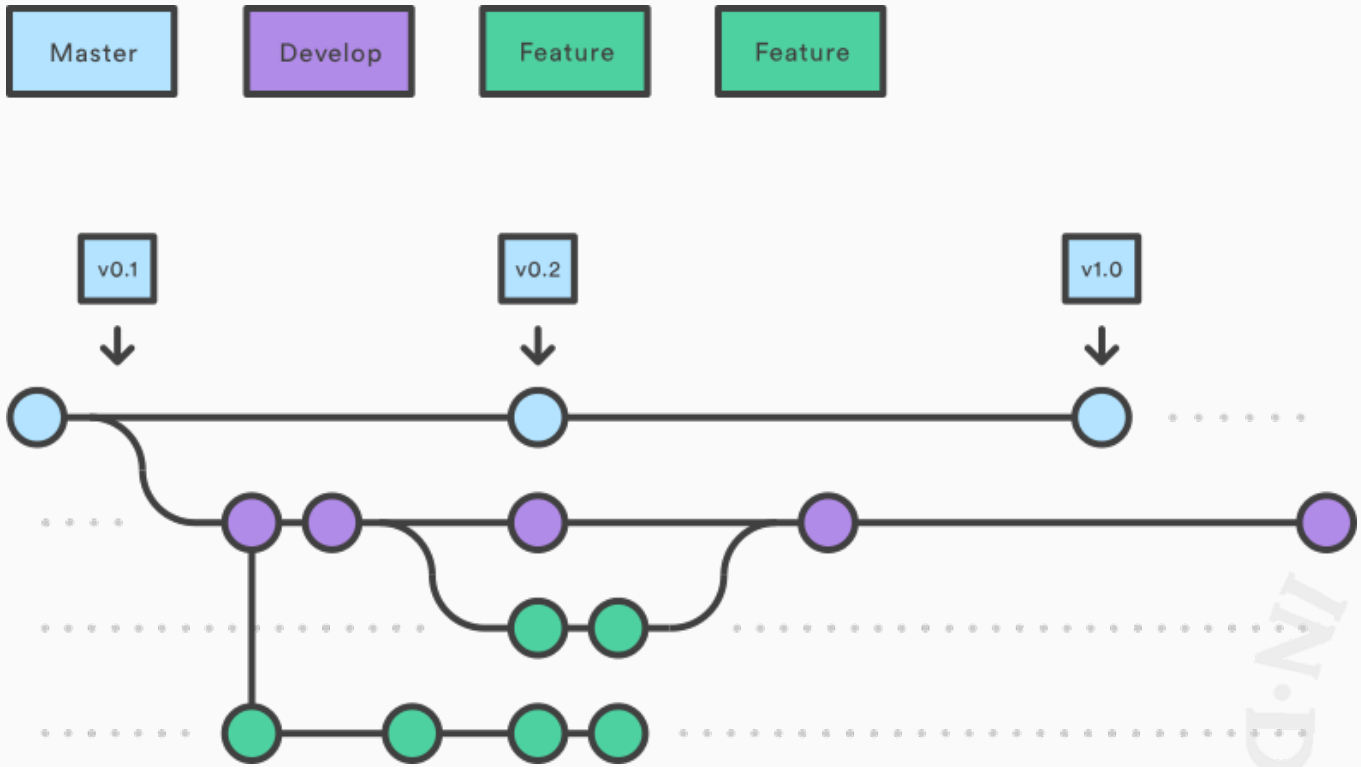
Gitflow Workflow



Develop and Master Branches.

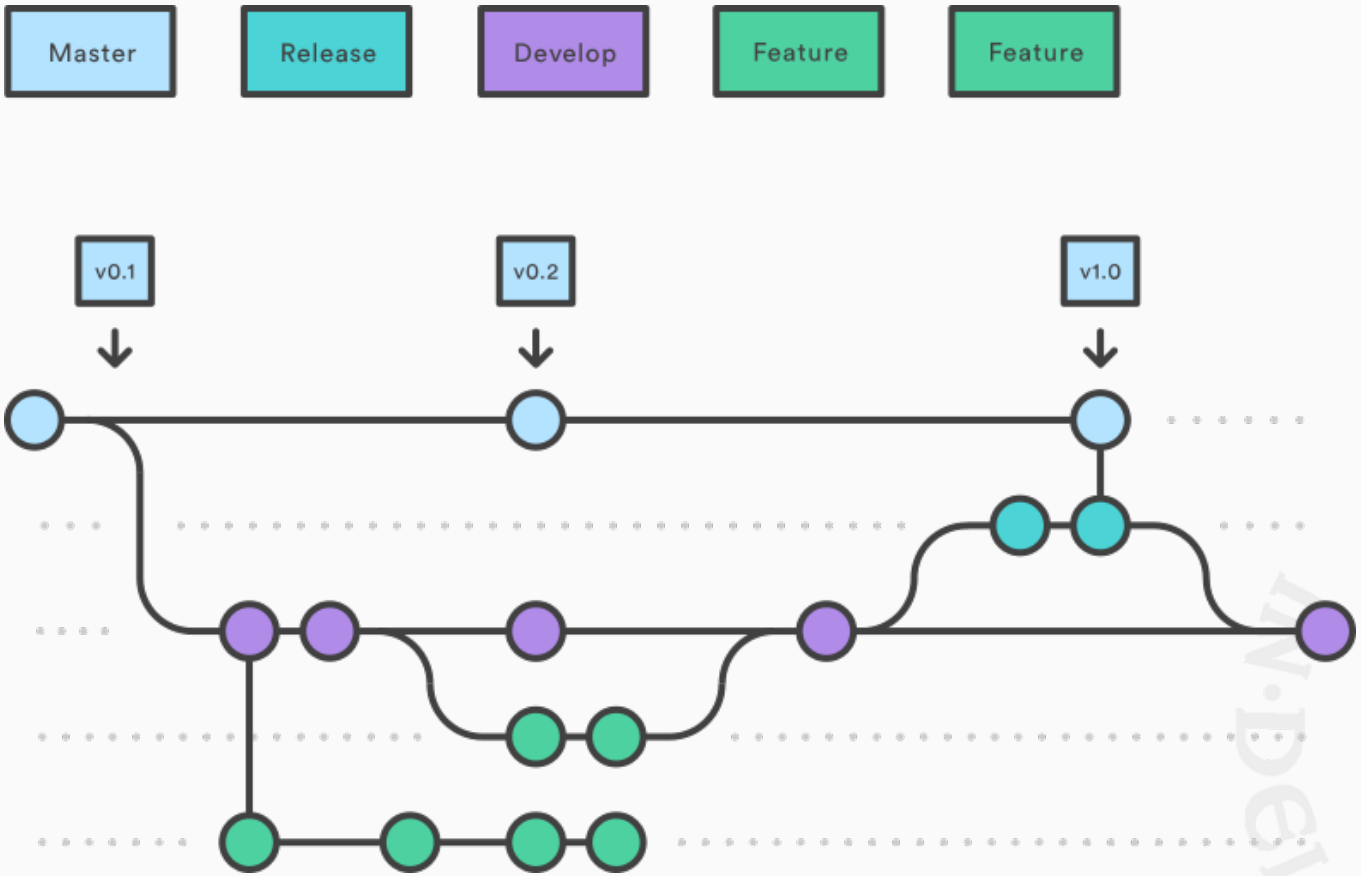
- Master branch only contains the minor and major versions.
  $\Longrightarrow$ e.g. *Debian Stable*
- Develop branch contains all the intermediate moditications.
  $\Longrightarrow$ e.g. *Debian Unstable*

Gitflow Workflow



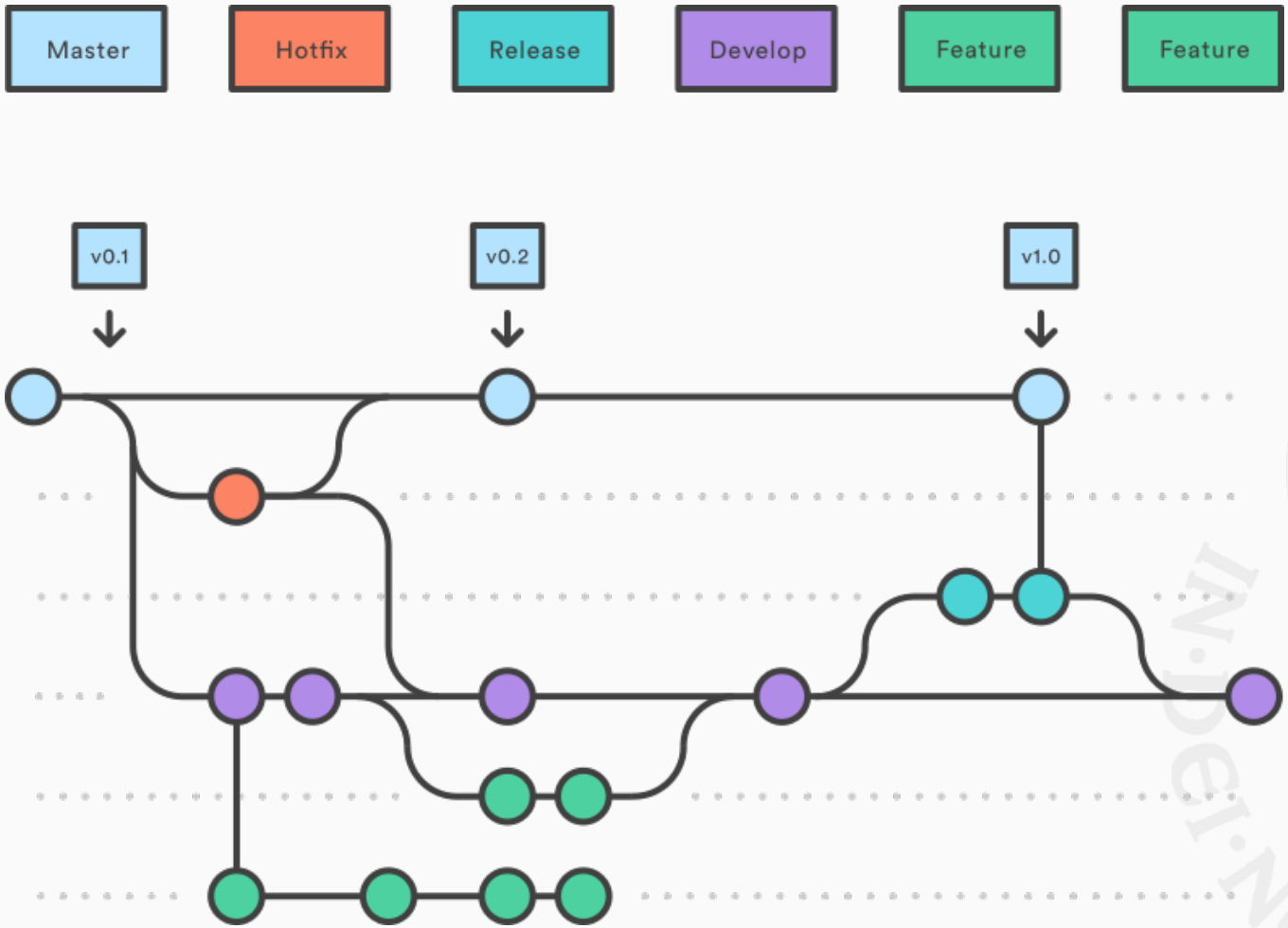
- Features are developped as branches of the Develop branch.
  $\Longrightarrow$ e.g. *Debian Experimental*

Gitflow Workflow



- A Release branch contains the "frozen" features.
  $\Longrightarrow$ e.g. *Debian Testing*

Gitflow Workflow



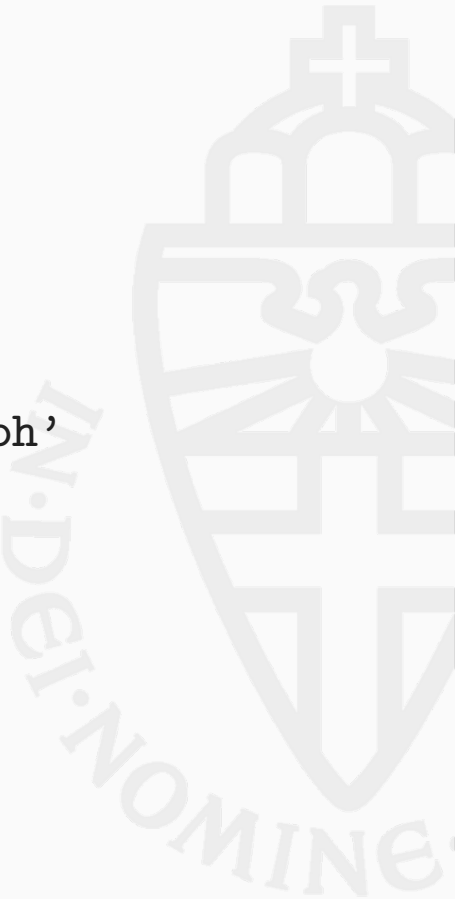
- if an issue in master is detected a hotfix branch is created from master

Bonus

bonus

In your `\.bashrc` or `\.zshrc`:

```
alias gtree='git log --oneline --decorate --all --graph'
```



Thank you.