Versioning solutions

Advanced Git

Thomas Domingues

Summary

- Introduction
- Branches
- Merge and Rebase
- Workflow - Git Flow
- Manage and edit Git History
  - Stash / Pop
  - Reset
  - Cherry-pick
- Practical work

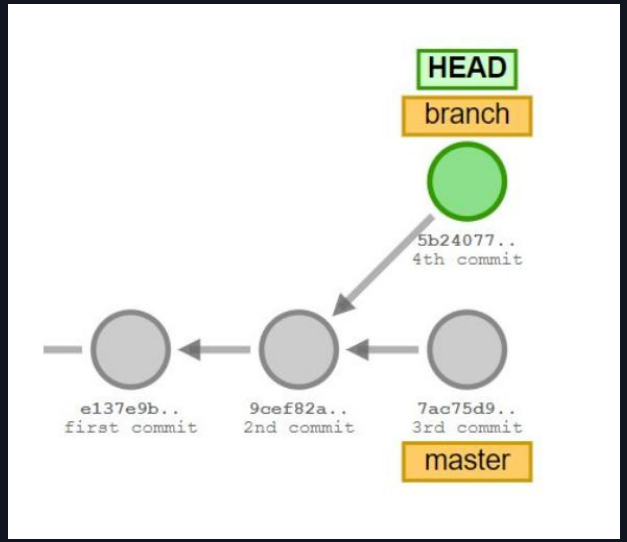
Introduction

Now you know how to basically work with git.

But how can I work efficiently with it?

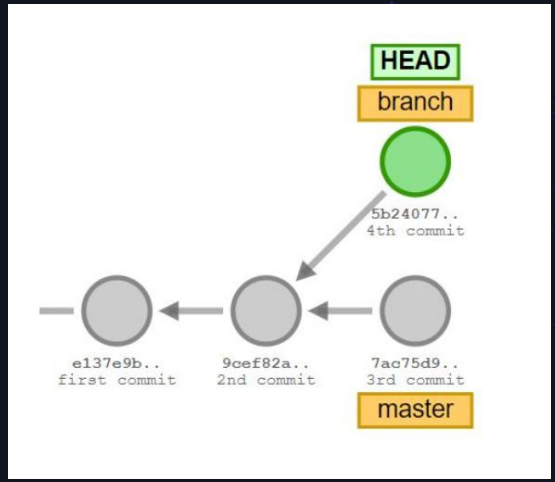
Branches

Branches’ main strength: Isolate two different developments



Branches

```
user@/git-project$ git commit -m “first commit”
user@/git-project$ git commit -m “2nd commit”
user@/git-project$ git commit -m “3rd commit”
user@/git-project$ git checkout HEAD~1
user@/git-project$ git checkout -b branch
user@/git-project$ git commit -m “4th commit”
```

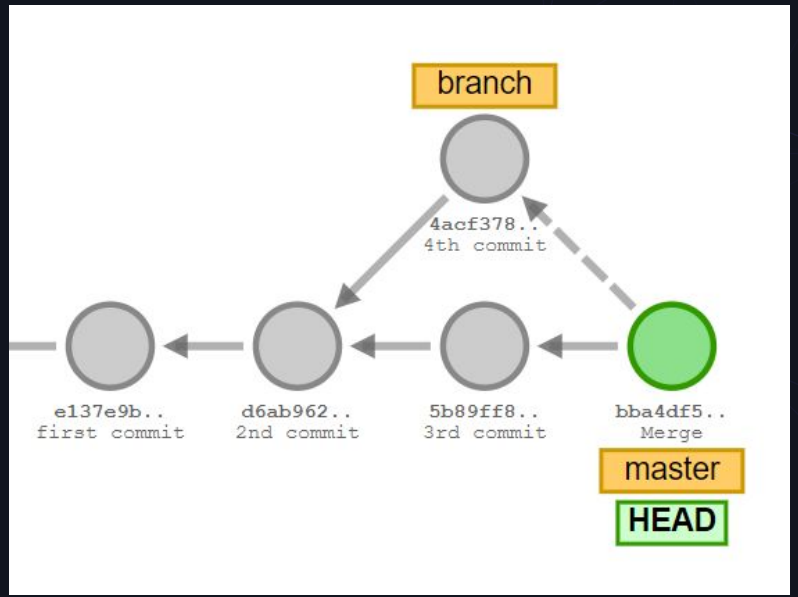


https://git-scm.com/docs/git-checkout & https://git-scm.com/docs/git-branch

Merge

Join two or more development histories together

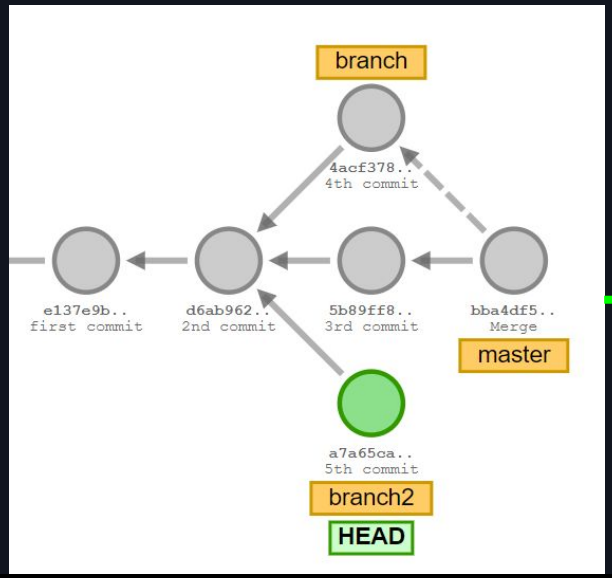
```
user@/git-project$ git checkout master
user@/git-project$ git merge branch
```

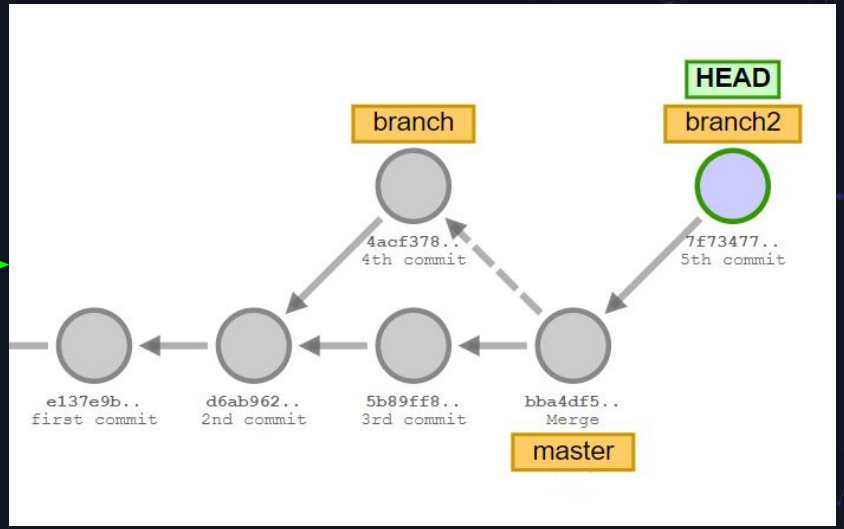


https://git-scm.com/docs/git-merge

Rebase

Update a branch by reapplying commits on top of another base tip



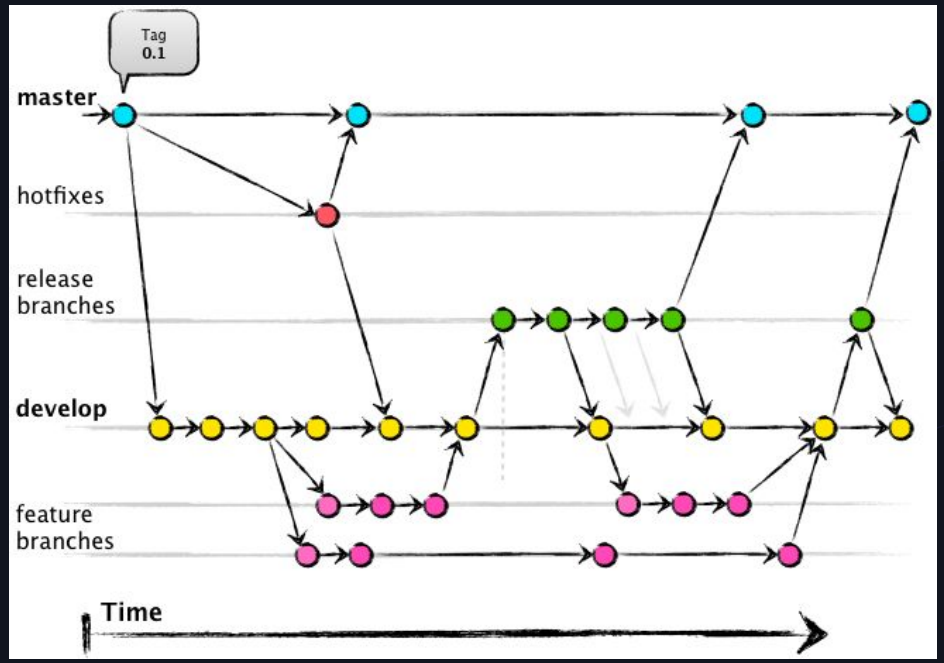



```
user@/git-project$ git checkout HEAD~2
user@/git-project$ git checkout -b branch2
user@/git-project$ git commit -m "5th commit"
```

```
user@/git-project$ git rebase master
```

https://git-scm.com/docs/git-rebase

Workflow - Git Flow



(Source)

Workflow - Git Flow

5 main branches:

- Master (Unique & Permanent): Stable, tested and validated code ready to prod
- Develop (Unique & Permanent, based on master): Code of next version of the application. May be unstable. Once a feature is done, code is merged in develop
- Feature (One per feature, based on develop): WIP Code implementing a new functionality
- Release (Unique, semi-permanent): Code being tested and patched for incoming new version. Once code is stable, it is merged to master
- Hotfix (One per hotfix): Bug fixes for master.

Workflow - Git Flow

Setup the Git Flow

```
user@/git-project$ git init
user@/git-project$ git commit -m “Init project”
user@/git-project$ git checkout -b develop
user@/git-project$ git checkout -b feat/readme
user@/git-project$ touch README.md && git add README.md
user@/git-project$ git commit -m “Add README.md
> ”
user@/git-project$ git checkout develop
user@/git-project$ git merge feat/readme
[...]
```

Manage and edit Git History

The second big strength of Git, after allowing version control, is to manage and edit previous commits, merges and other interactions on the project. Let’s see some commands

Git Stash

Stash the changes in a dirty working directory away

```
user@/git-project$ git stash
Saved working directory and index state WIP on master: ef11547 Init project
user@/git-project$ git stash list
stash@{0}: WIP on master: ef11547 Init project
```

https://git-scm.com/docs/git-stash

Git Stash Pop

Remove a single stashed state from the stash list and apply it

```
user@/git-project$ git stash list
stash@{0}: WIP on master: ef11547 Init project
user@/git-project$ git stash pop 0
[...]
Dropped refs/stash@{0}
```

https://git-scm.com/docs/git-stash

Git Reset

Reset current HEAD to a specified state:

- `git reset HEAD~1` - Reset HEAD to the previous commit
- `git reset ef11547` - Reset HEAD to the specific commit

https://git-scm.com/docs/git-reset

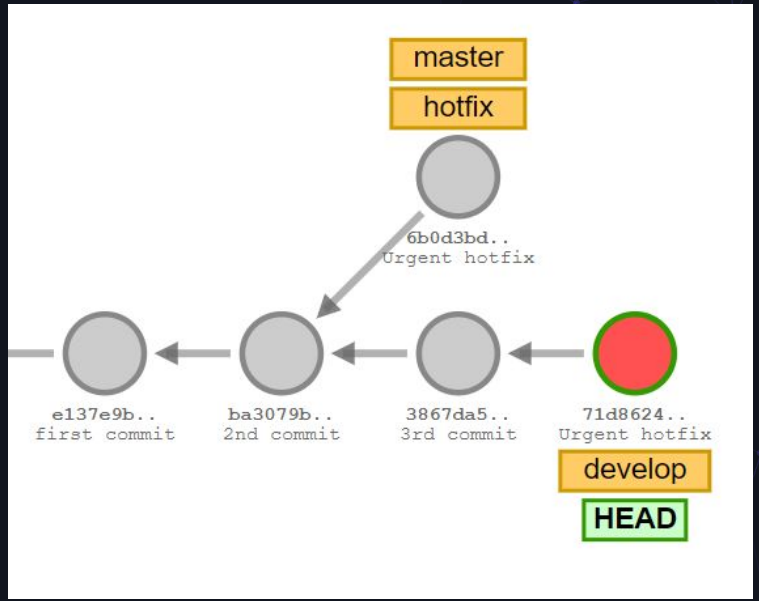
Git Cherry-pick

Copy a specific commit (identified by the hash) and apply it on top of the current branch

```
user@/git-project$ git checkout -b hotfix
user@/git-project$ git commit -m “Urgent hotfix”
user@/git-project$ git checkout master
user@/git-project$ git merge hotfix
user@/git-project$ git cherry-pick 6b0d3bd
```

(Proper solution here is not to cherry-pick but merge. As you can see, a new commit (and hash) is generated that will cause conflicts)

https://git-scm.com/docs/git-cherry-pick



Practical work

- Go to https://learngitbranching.js.org/?locale=en_US
- Do all main exercises except Advanced Topics (bonus)