Git Workflows and Gitlab

Micael Oliveira and Martin Lüders

Octopus Course 2023, MPSD Hamburg

Revisiting Git

Commits

What is included in a commit?

- A snapshot of the sources
- A timestamp
- A log message
- Zero or more parent commits

Two commits whose content differ in any way are **different** commits!

Note 1: A merge commit has two or more parent commits.
Note 2: The initial commit has no parent commit.

Branches and tags

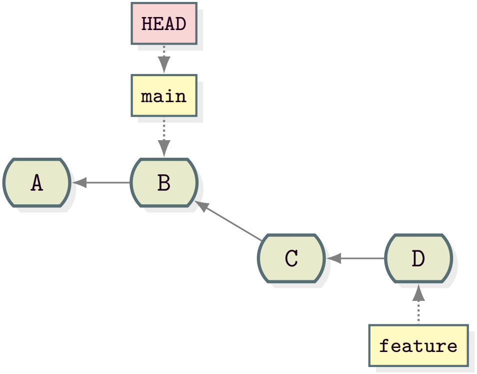
- A branch can be viewed as a pointer to a commit
- A tag is an immutable pointer to a commit
- Branches and tags are often interchangeable!

Example:

```
$ git branch foo bar
```

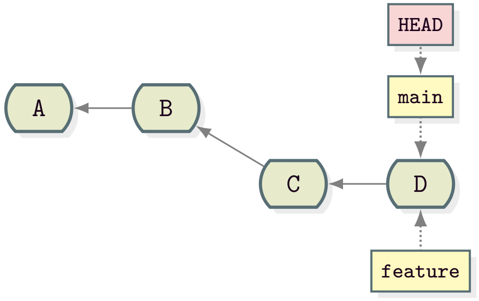
- Creates a new branch named `foo` using `bar` as a starting point.
- `bar` can be any commit-ish object: branch, tag, commit, etc.

Fast-forward merges



```
$ git merge feature
```

Fast-forward merges



- No merge commit is created
- Use `git merge --no-ff` to force creation of merge commit

Git Workflows

Workflows

Why?

- Git is very flexible and powerfull
- Workflows are recipes and recommendations to use git in a consistent way
- They are necessary to develop code in a collaborative way
- The are necessary to prepare releases and hotfixes

Workflows

Good workflows:

- Scale with the number of developers
- Do not impose any large overhead
- Prevent mistakes or allow to easily fix them

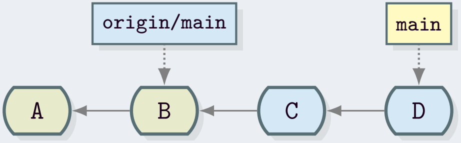
Prelude

Some assumptions in the following examples:

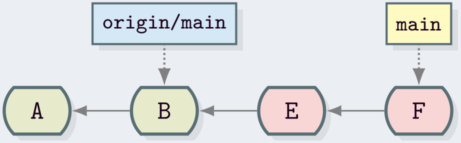
- There is always a remote repository that represents the official project
- Each developer has a local repository
- Unless otherwise stated, local repositories are clones of the official repository

Centralized workflow

Anne



Bob

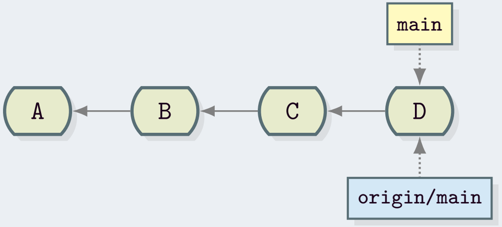


Centralized workflow

- Changes are published by pushing them to the official repository

Anne

```
$ git push origin main
```

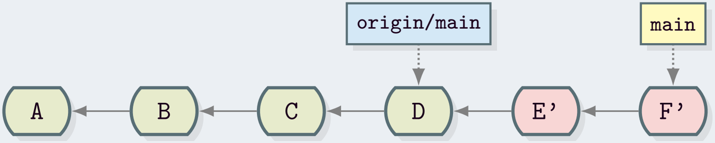


Centralized workflow

- Bob needs to get Anne's changes before publishing his own changes
- A rebase is necessary, otherwise the push would fail

Bob

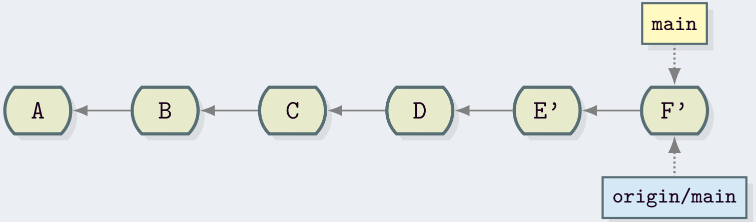
```
$ git pull --rebase
```



Centralized workflow

Bob

```
$ git push origin main
```

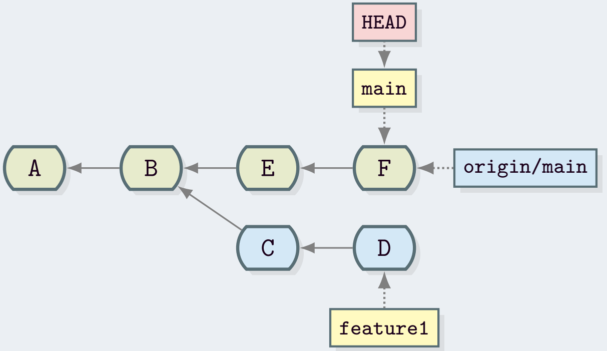


- Each developer can only work on one feature at a time

Feature branch workflow

- Feature branches are branched from main

Anne

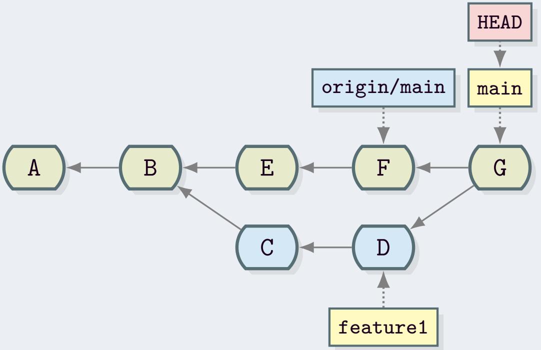


Feature branch workflow

- Feature branches are merged into main

Anne

```
$ git merge feature1
```

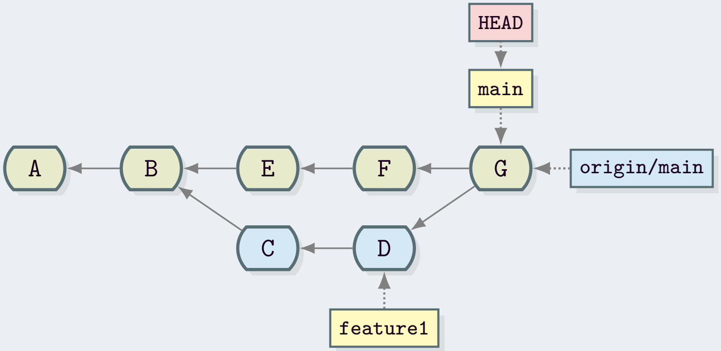


Feature branch workflow

- The result must be published to the official repository

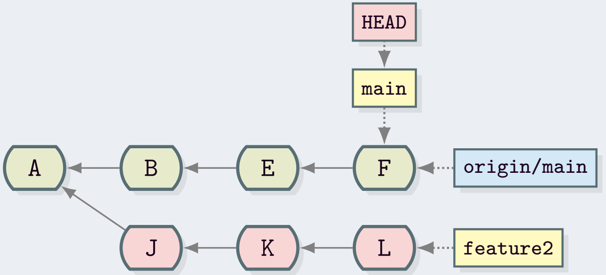
Anne

```
$ git push
```



Feature branch workflow

Bob

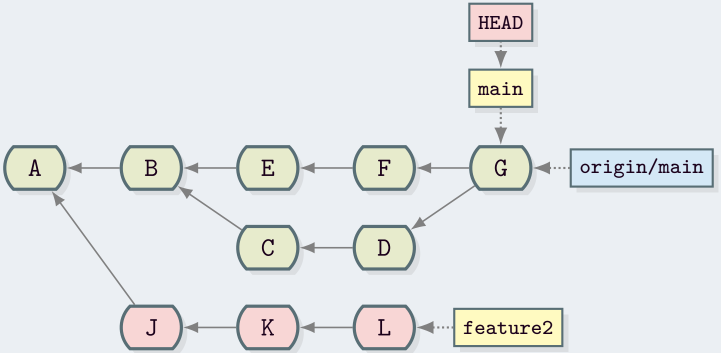


Feature branch workflow

- Always pull latest commits to main before merging!

Bob

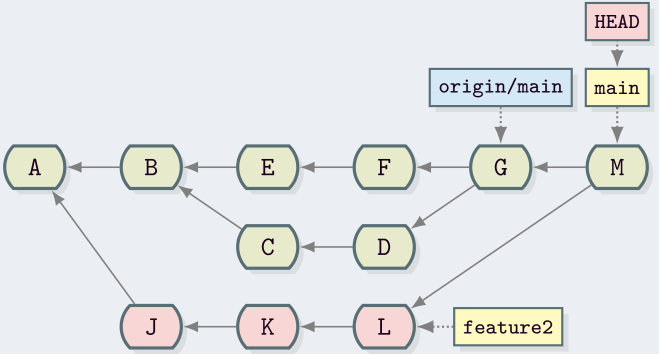
```
$ git pull
```



Feature branch workflow

Bob

```
$ git merge feature2
```



Feature branch workflow

- Work on different features is independent
- Feature branches should be short lived

Forking workflow

- Several web applications like GitHub and GitLab provide forks
- A fork is a server-side copy of the official repository
- A mechanism is provided to merge a branch from the fork into the main repository (pull/merge request)
- Contributors do not need to have write permissions to the official repository
- Can be used with other workflows that use feature branches

Releases and hotfixes

Interlude: Releases and Hotfixes

- Some software is periodically released for production: production release
- A production release usually includes new features and bugfixes
- Production releases are usually less frequent than the addition of new features (exception: continuous delivery)
- Hotfixes are releases that include only critical bugfixes
- Hotfix releases should not include new features

Interlude: Releases and Hotfixes

How to do releases with git?

- Tags are used to mark releases
- Workflow needs to incorporate some procedure to create the releases and the hotfixes
- Suitable procedure depends on several things. For example:
  - How often one does a new release
  - Is a new production release based on the previous release?
  - How many simultaneous releases are maintained?

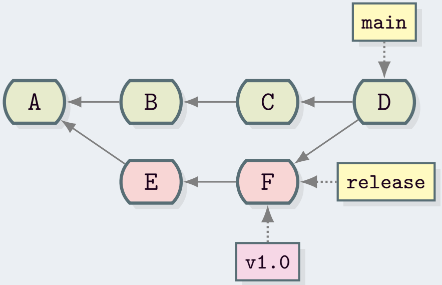
OneFlow

- Alternative to GitFlow
- Only one long-lived branch (main)
- Same support branches as GitFlow
- Feature branches are branched from and merged into main

`www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow`

OneFlow

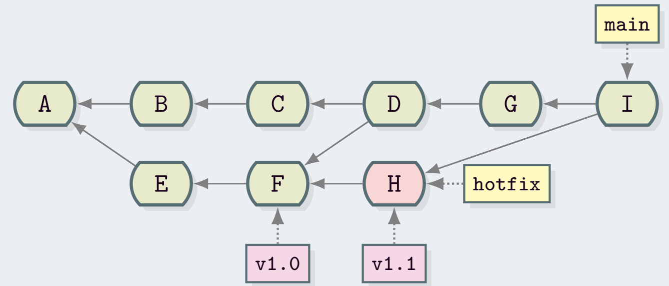
Release branches



- Release branch is branched from and merged into main
- Last commit of release branch is tagged

OneFlow

Hotfix branches



- Hotfix branch is branched from last release tag
- Hotfix branch is merged into main
- Last commit of hotfix branch is tagged

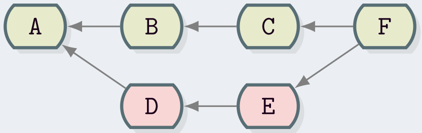
OneFlow

- Simpler than GitFlow
- Less merges than GitFlow
- It is able to do all that GitFlow can do
- It is possible to do hotfixes from older releases (exercise: think how this would work)

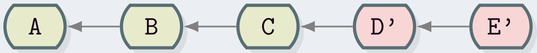
Merge vs Rebase

Interlude II: Merge vs Rebase

Merge


$$\begin{tikzcd} A & B \arrow[l] & C \arrow[l] & F \arrow[l] \arrow[dl] \\ & D \arrow[ul] & E \arrow[l] & \end{tikzcd}$$

Rebase


$$\begin{tikzcd} A & B \arrow[l] & C \arrow[l] & D' \arrow[l] & E' \arrow[l] \end{tikzcd}$$

Merge:

- Preserves history
- Easy to revert/reset

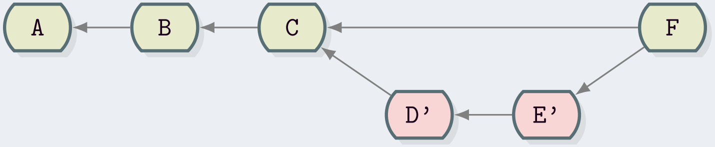
Rebase:

- Linear history
- Opportunity to clean up the branch history

Interlude II: Merge vs Rebase

- It's a matter of taste!
- Should the git history reflect the "real" development history?

Compromise: only merge if fast-forward was possible



- A rebase is usually needed before merging
- GitLab can enforce this

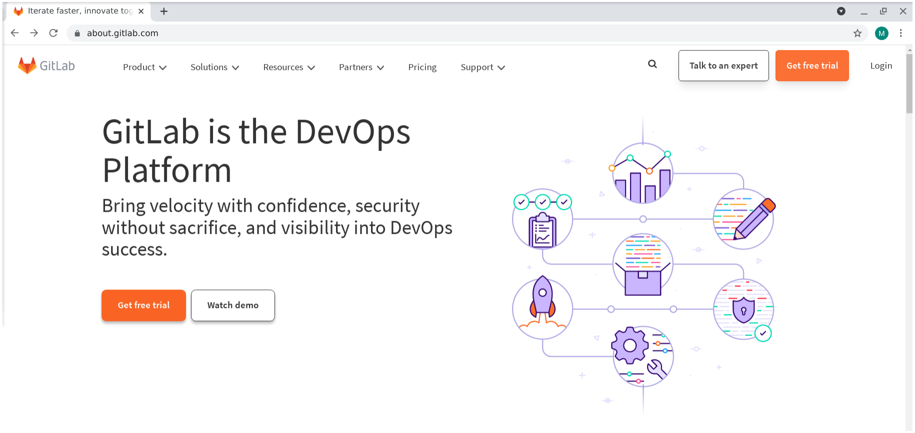
GitLab

Gitlab

- Hosting service for Git repositories
- Provides forks and merge requests
- Issues: bug reports, feature requests, project tasks, etc
- Provides its own CI service (GitLab CI)
- Other CI services can be used through the public API
- Open source
- Many more features...

Gitlab

Guided tour



GitLab is the DevOps Platform

Bring velocity with confidence, security without sacrifice, and visibility into DevOps success.