Advanced git features and git workflows

Christoph Niethammer

20. October 2023

git overview

- Git is a distributed version control system
- Widely adopted in software development
- Often used together with software development plaforms, e.g. github, gitlab, …, **code.hlrs.de**

Goals of this talk

- Get better understanding of git concepts
- Learn advanced git commands
- Learn common workflows for personal and collaborative work
- How to use git with a software development platform

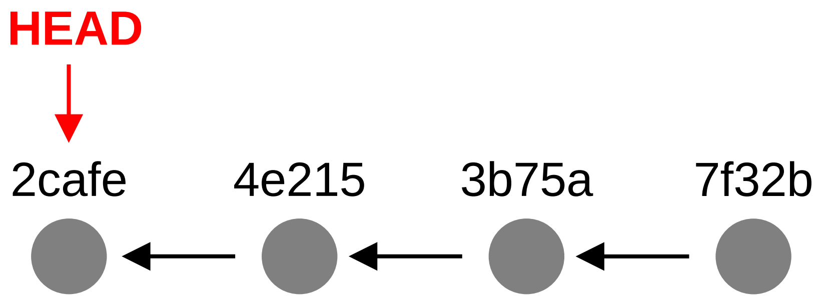
What you should already know…

Basic git commands:

- **init, clone**
  → setting up a basic git repository
- **config**
  → configuring your user.email/name
- **add, commit**
  → adding files and commiting to your local repository
- **status, diff, log**
  → inspecting work and history
- **pull, push**
  → perform basic sync/update of local and remote repository

git concepts and commands

commit



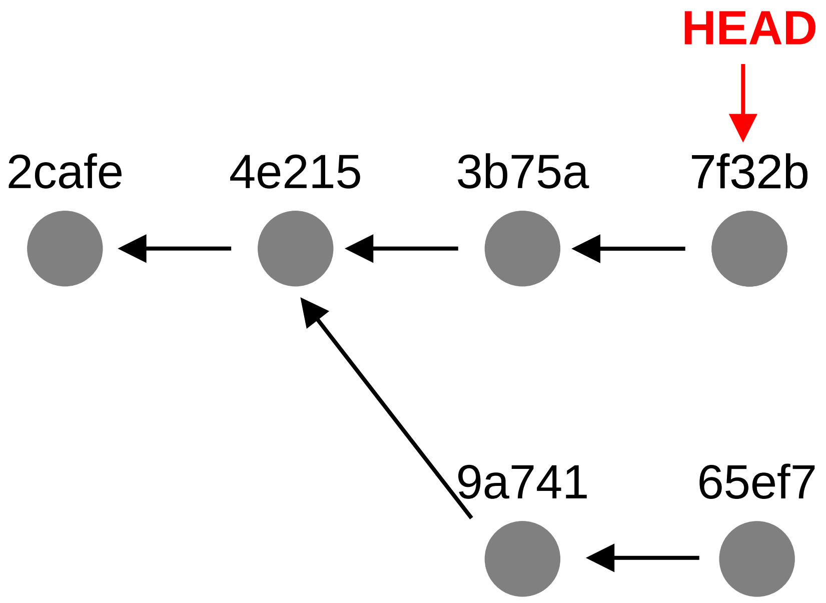
A commit

Tracks files:

- Paths
- Permissions
- Contents

+ Commit message

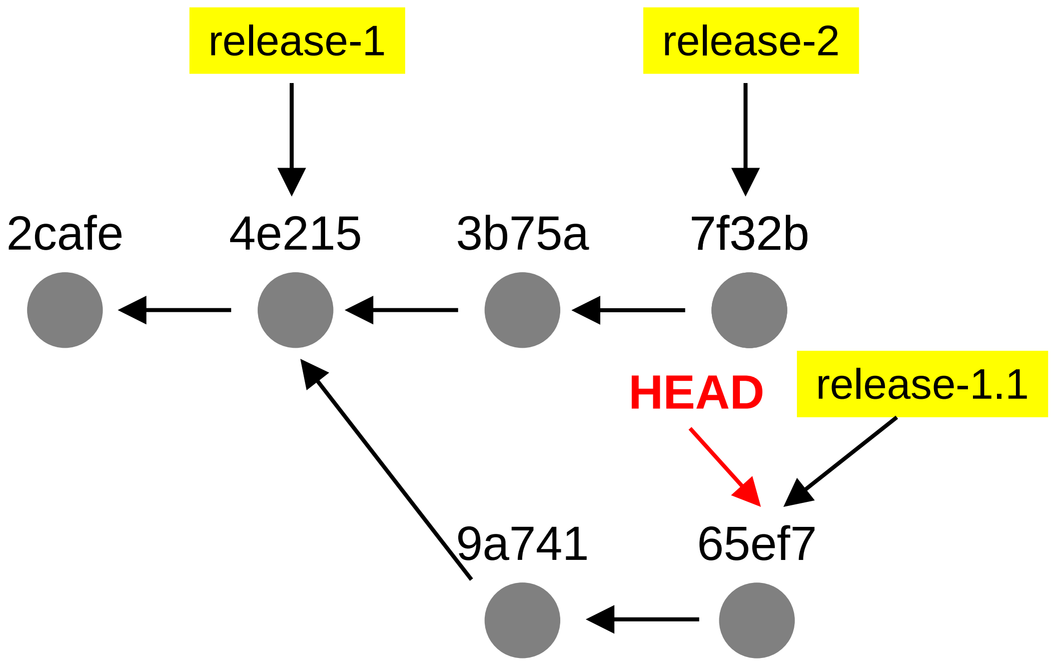
checkout



checkout:

- ‘update HEAD’
- update files in working tree

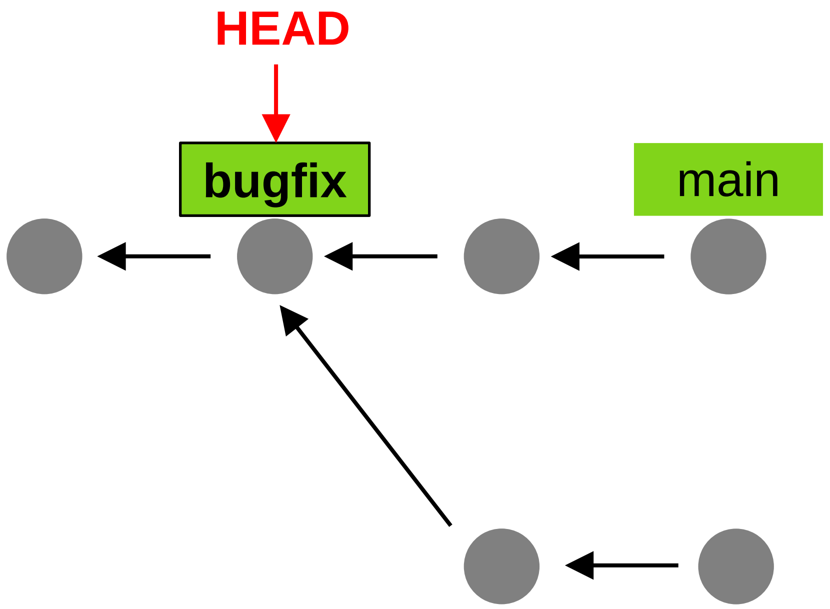
tag



tag:

- named pointer to a specific commit
- two types:
  - lightweight
  - annotated

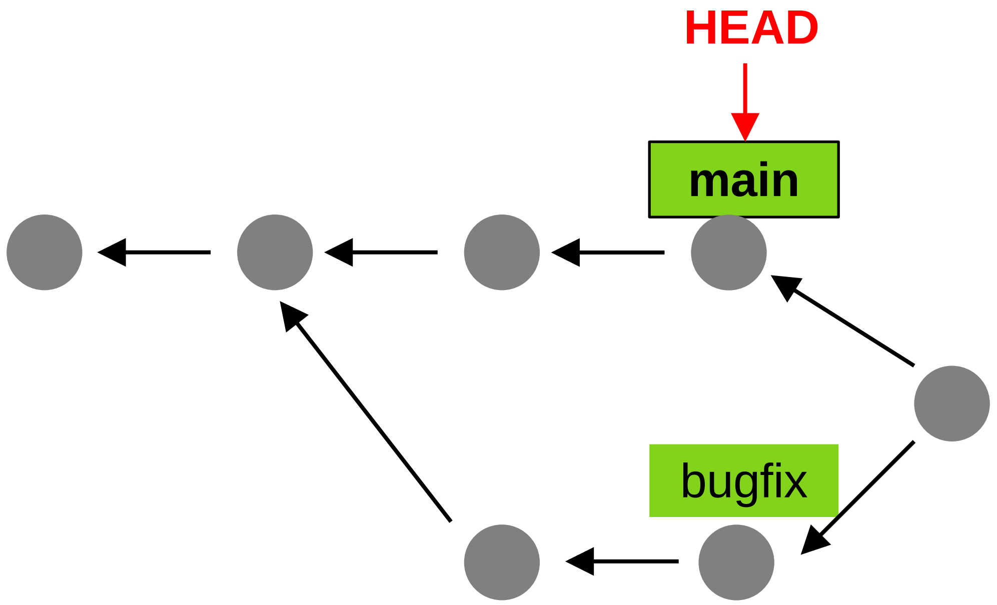
branch



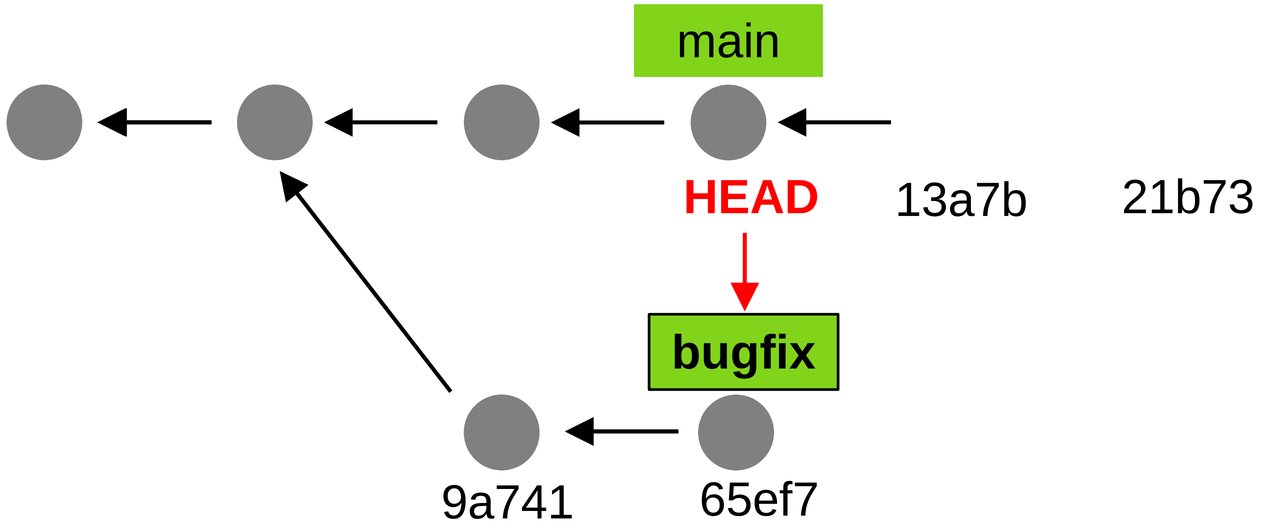
branch:

- ‘automatic movable label’

merge



rebase



rebase:

- applies set of changes on top of other commit

Do not rebase published branches!

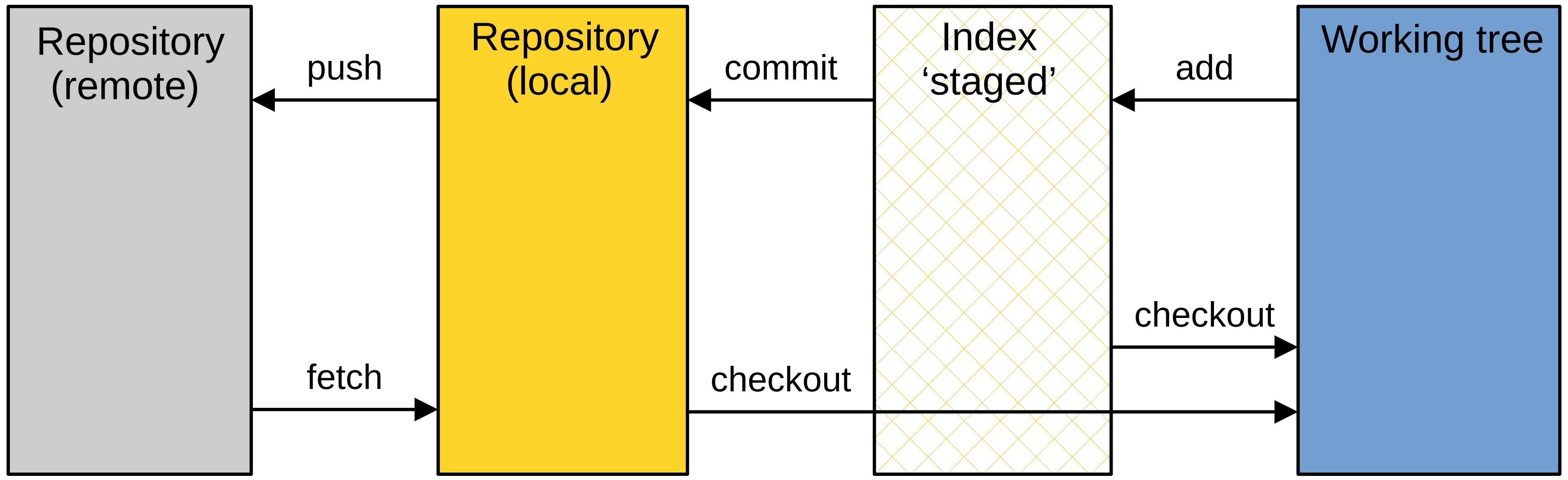
Interactive rebase

Used to restructure commits (before pushing):

- change commit order
- squash multiple commits into one
- drop unrelated commits
- change commit messages

Interactive rebase - Demo

Working tree, Index, and Repository



Interactive staging

- partition larger changes into several focused commits
- Possible granularity:
  - file level
  - patch level
- try `git gui`

Interactive staging - Demo

log

Default: chronlogic list of commits

- log --graph
- log A ^B
- log --follow file/path
- log -G<regex>

git log - Demo

Writing good commit messages

- **Subject** (~50 characters):
  - descriptive!
- **Description** (go in detail)
  - context information
  - what was changed?
- **Trailer**
  - Signed-off-by:
  - Co-authored-by:

Examples:

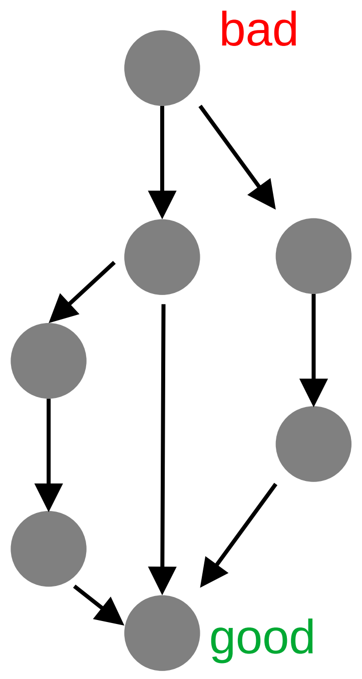
```
Fix bug in program initalisation

The initalisation routine was slow. Therefore,
changed algorithm from A1 to A2.
Implementation based on Ref2.

Signed-off-by: Hello Me <hello.me@mail.com>
```

```
Chapter 6: added related work
```

bisect

“When was this bug introduced?“

1) Find commits with/without the bug
2) That’s it – just follow git’s instructions :)



git bisect - Demo

git workflows

Git workflows

To be considered when chosing a workflow:

- Project requirements
- Team structure
- Development practices

Centralized workflow

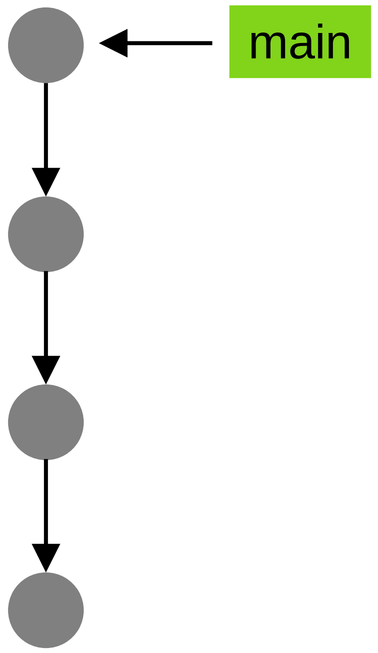
Single centralized repository with single main branch

Pros:

- simple

Cons:

- frequent merge conflicts
- unstable main branch



Feature branch workflow

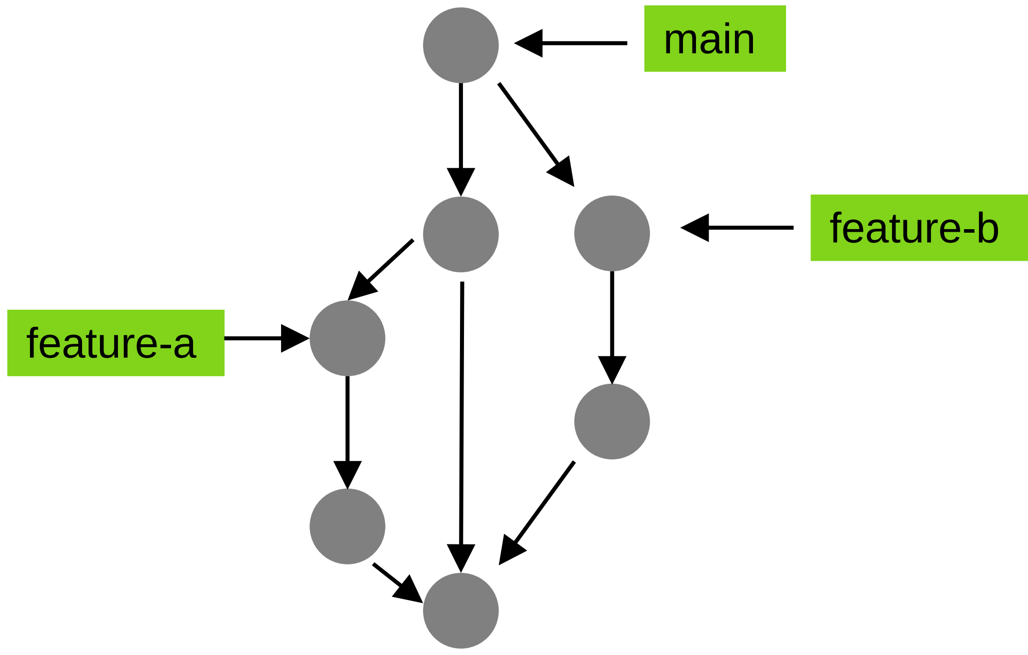
New branch for each feature based on main branch. Merge back into main branch once completed.

Pros:

- parallel development
- simplified testing and review process

Cons:

- merge conflicts often complicated
- slower integration



Gitflow workflow

Involves multiple branches to manage different aspects of the development process:

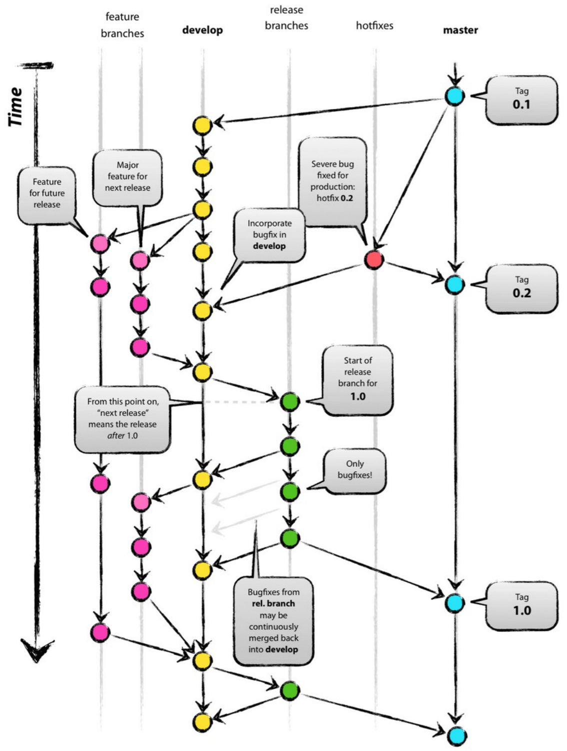
- main: production ready code
- develop: integration point
- feature: individual feature development
- release: preparation for production release
- hotfix: critical issues in production code

Pros:

- clear separation of concerns
- parallel development

Cons:

- complex
- often slower release cycle


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Source: https://nvie.com/posts/a-successful-git-branching-model/

Release branch workflow

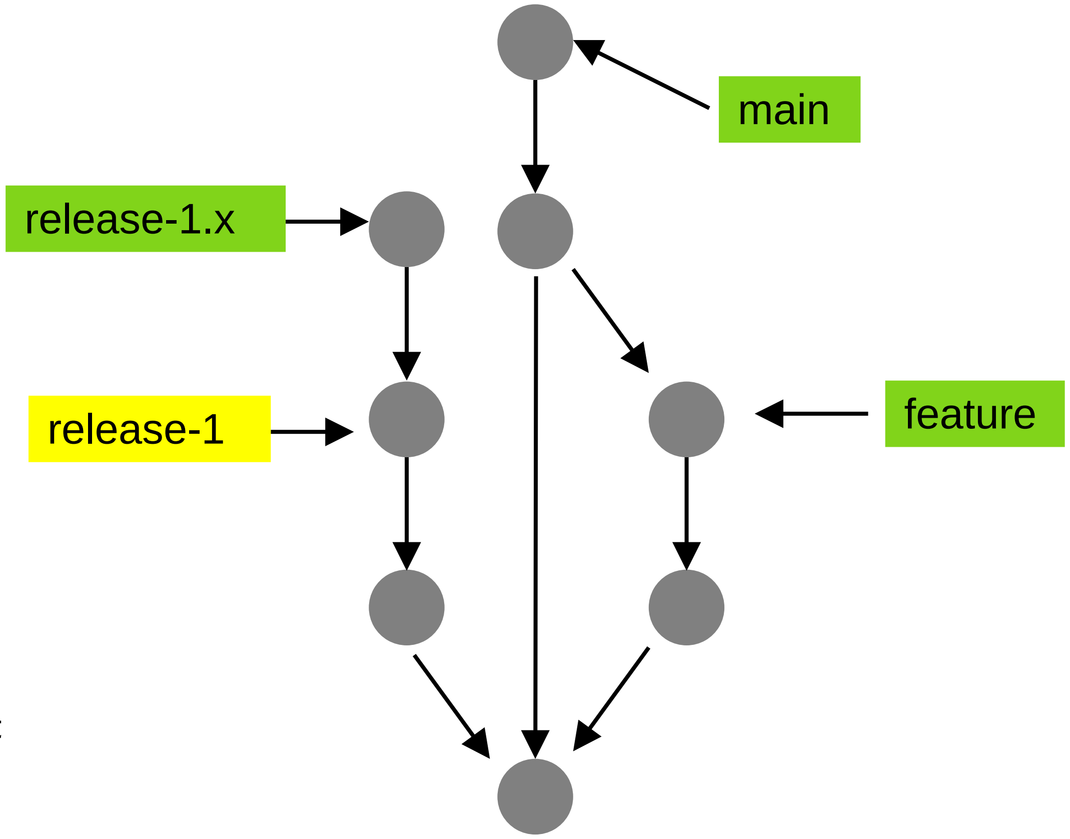
Branch of main for each release. Release branches only for stabilisation. New features go into main branch – possibly combine with feature branche workflow for stability.

Pros:

- flexible release management
- clear separation between development and release

Cons:

- increased complexity with potentially overlapping development cycles
- potential delays in development due to resource split



code.hlrs.de / Github & Co

Gitea (code.hlrs.de)

Software development and version control platform

- structured using projects and teams
- provides issue tracker, wiki, milestones, etc.
- hosts git repositories (central + private)
- manages pull (merge) requests for distributed git workflows

- can be used for other purposes than software ;)

Issues and pull requests

Issue:

Tracker to document bug reports, feature requests, etc.

Pull (merge) request:

request to merge a specific git branch into another one (both branches are in the git repositories on the gitea server)

Steps of a Pull request (workflow)

1) Clone public repository (e.g. code.hlrs.de)
2) Create fearure branch and make changes
3) Push feature branch to your public user repository
4) Create Pull request for the feature branch to be merge into the main branch of ther original repository
5) Wait for code reviews and approval
6) Merge by yourself or “gatekeeper“ (depending on repository policy)

Advanced git features and git workflows

Christoph Niethammer

20. October 2023

Thank you for your attention!