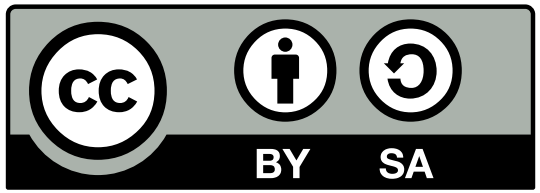
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Version Control Systems (VCS): Introduction to Git

J.-C. Bach

SIT211 – – 2025-2026

Work under Creative Commons BY-SA license



You are **free**:

- to **use**, to **copy**, to **distribute** and to **transmit** this creation to the public;
- to **adapt** this work.

Under the following terms:

- **Attribution** (**BY**): you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **ShareAlike** (**SA**): if you modify, transform, alter, adapt or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

See `http://creativecommons.org/licenses/by-sa/2.0/`

Important notes

- If you do not understand something, please ask your questions.
  We cannot answer the questions you do not ask
- If you disagree with us, please say it (politely)
- People don't learn computer science by only reading few academic slides: practicing is fundamental

Progress

Sentences one would have preferred not to hear

- *Aaaaah! Three months of work lost!*
- *Oops Was this file really important?*
- *Great, everyone has finished! Who integrates all the parts?*
- *Why did I wrote this piece of code?*
- *Great functionality, but I think the last week version was better. Uh which one?*
- *I cannot find the version we have made 6 years ago for BigCustomer Inc., I need it immediately for a new contract!*
- *I have already done this bugfix on my laptop I left at home.*
- *It doesn't work anymore! Who messed up my code?*

Motivations

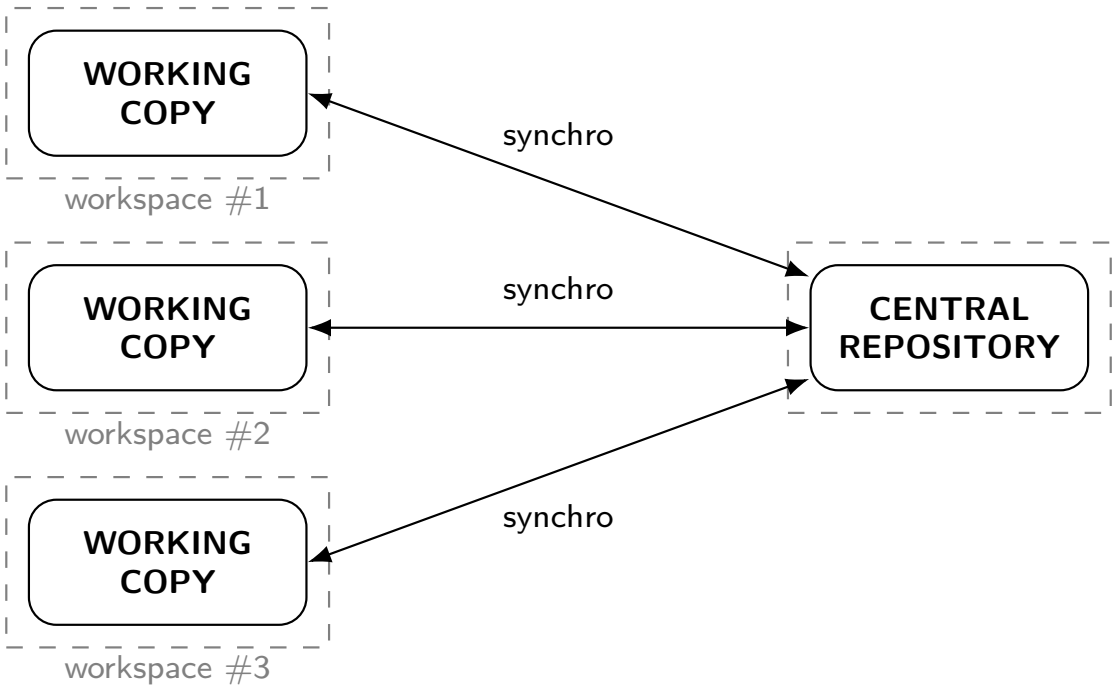
- Software traceability: tracking and documenting changes, retrieving former versions
- Flexibility: feature trials, quick rollbacks
- Parallelism and team work: multi-sites, multi-computers, multi-developers and multi-activities
- Safety: “backup” with history
  - though VCS are not (space) efficient backup systems

  $\Rightarrow$ One needs tools to solve these problems

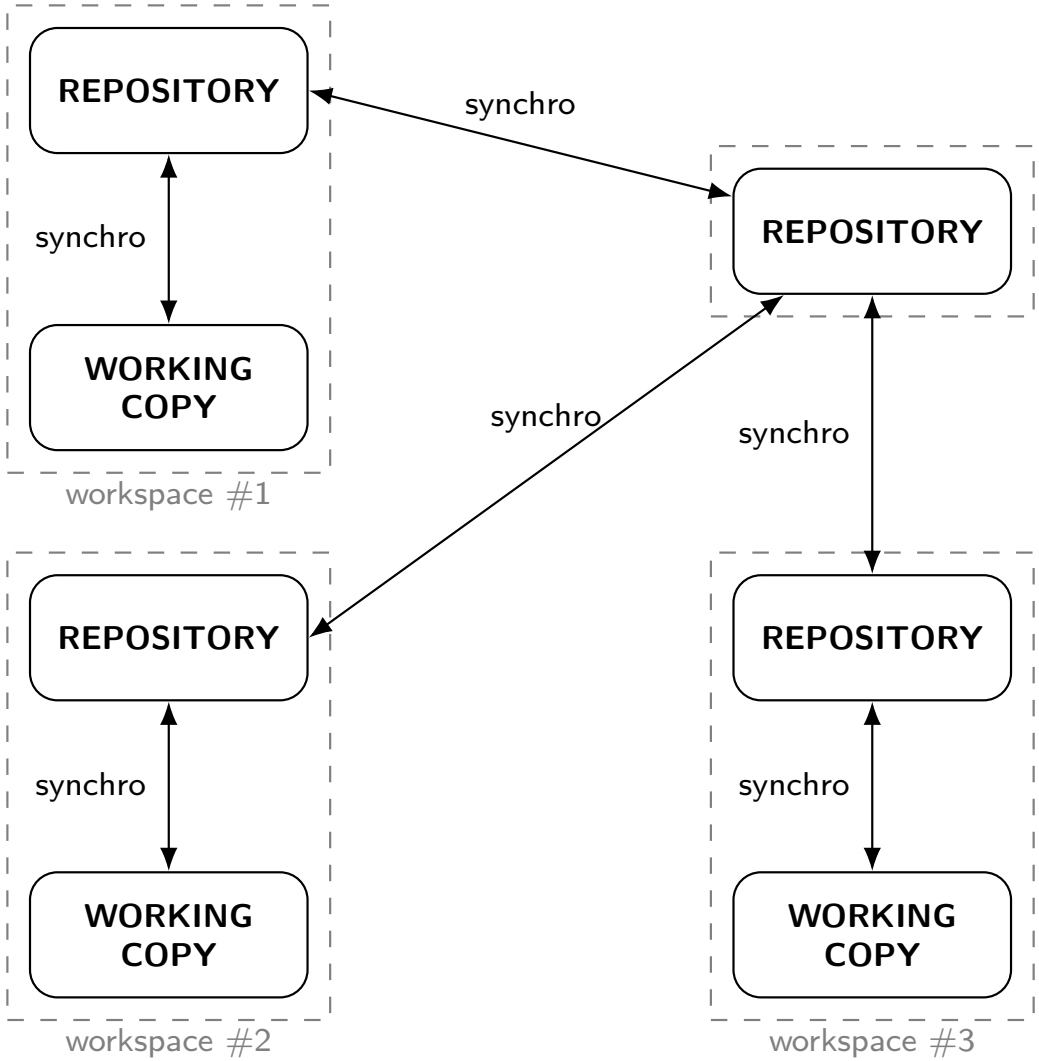
Version Control Systems (VCS)

- Used for
  - storing files
  - keeping track of changes on those tracked files
  - sharing
- Each collaborator works on a local copy
- Synchronization with one (or several) remote server(s)
- 2 families of VCS
  - centralised (Subversion, CVS,)
  - distributed (Git, Mercurial, Darcs,)

Architecture of a centralised VCS



Architecture of a distributed VCS



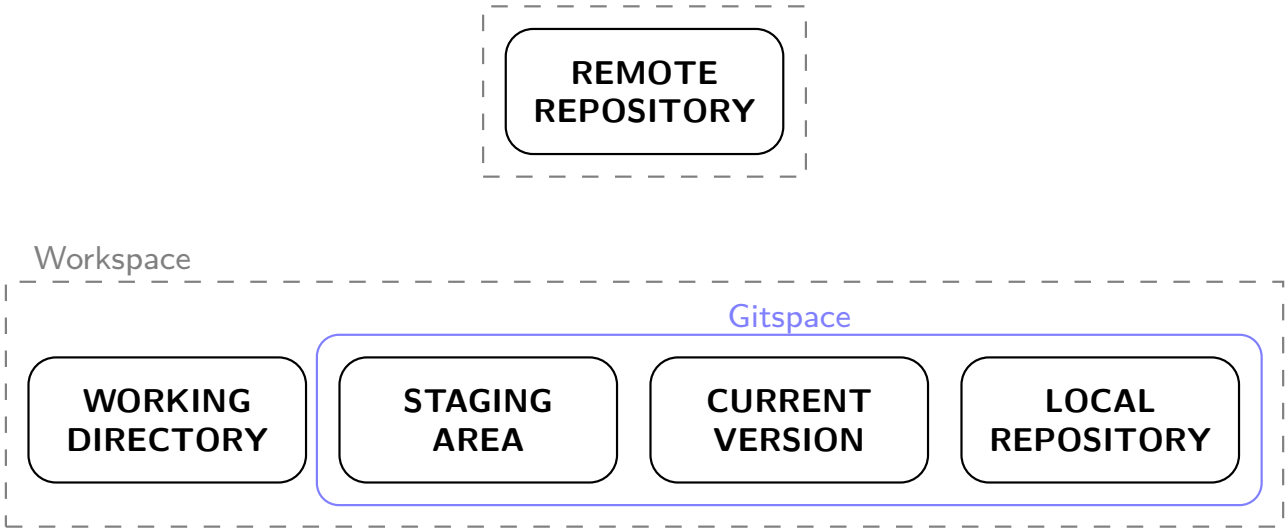
Focus on a specific VCS: Git

Why Git?

- very popular
- many platforms provide Git-based services (Bitbucket, Codeberg, Gitlab, Gitea, GitHub, SourceHut, ...)
- a bit less intuitive than other VCS for beginners, therefore if you are able to use Git, you will be able to use other VCS
- and because we had to choose a tool

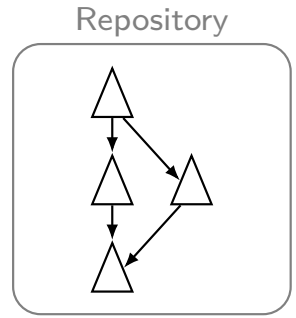
Progress

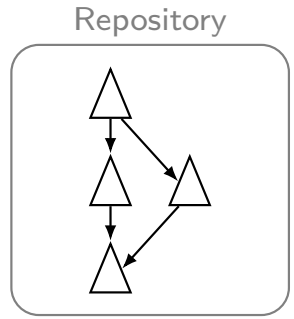
Git architecture and vocabulary



- working directory = files where changes are made
- staging area = current selected changes
- current version = current reference version
- (remote/local) repository = a database of changes

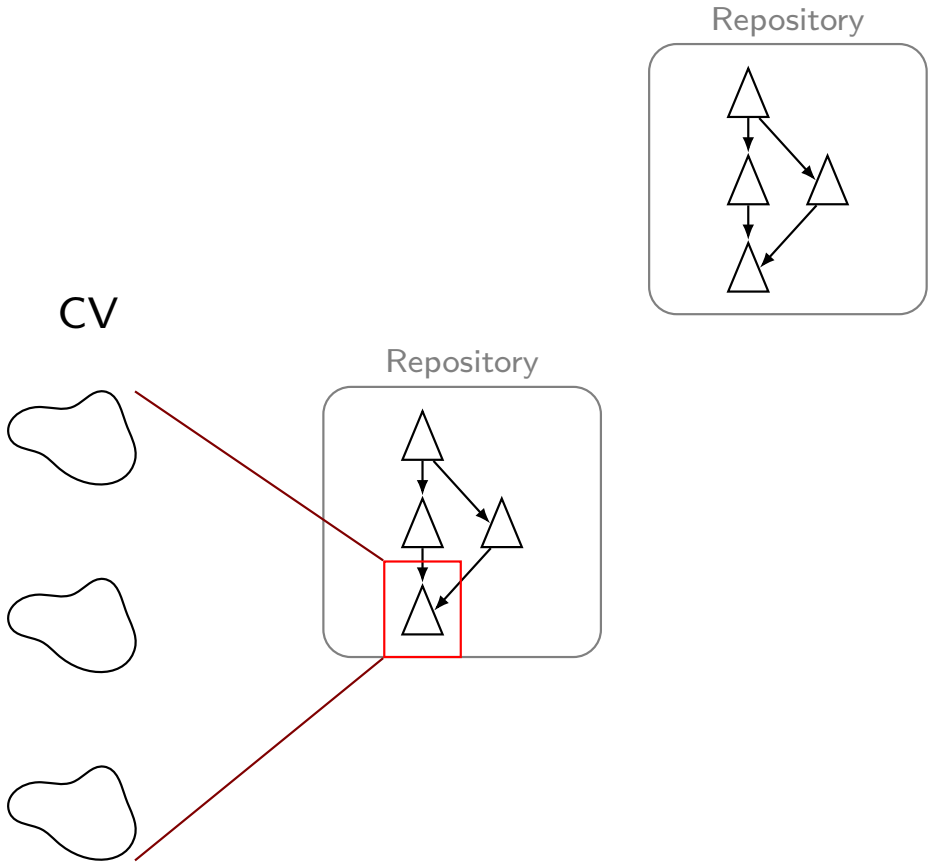
Diving into Git core





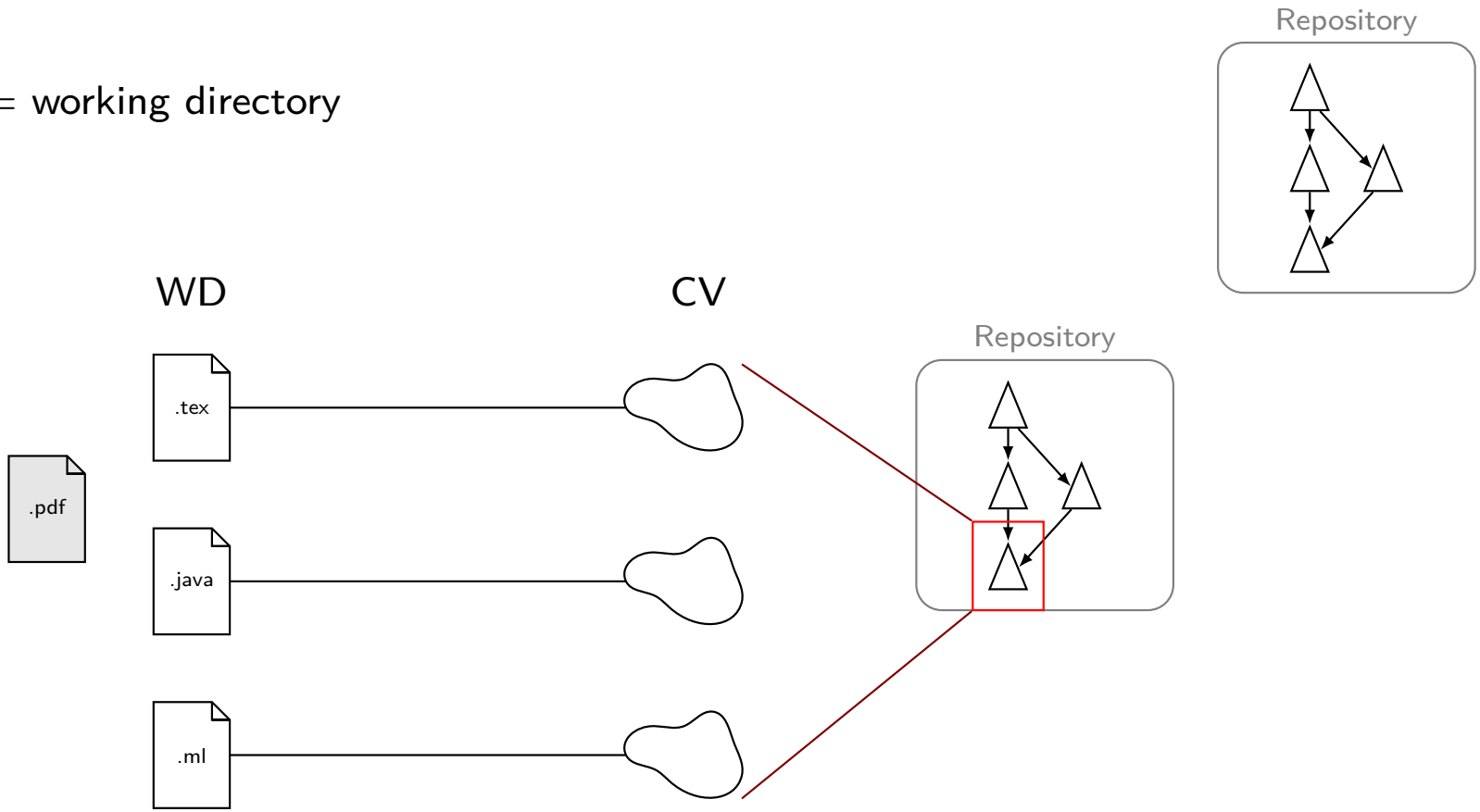
Diving into Git core

CV = current version

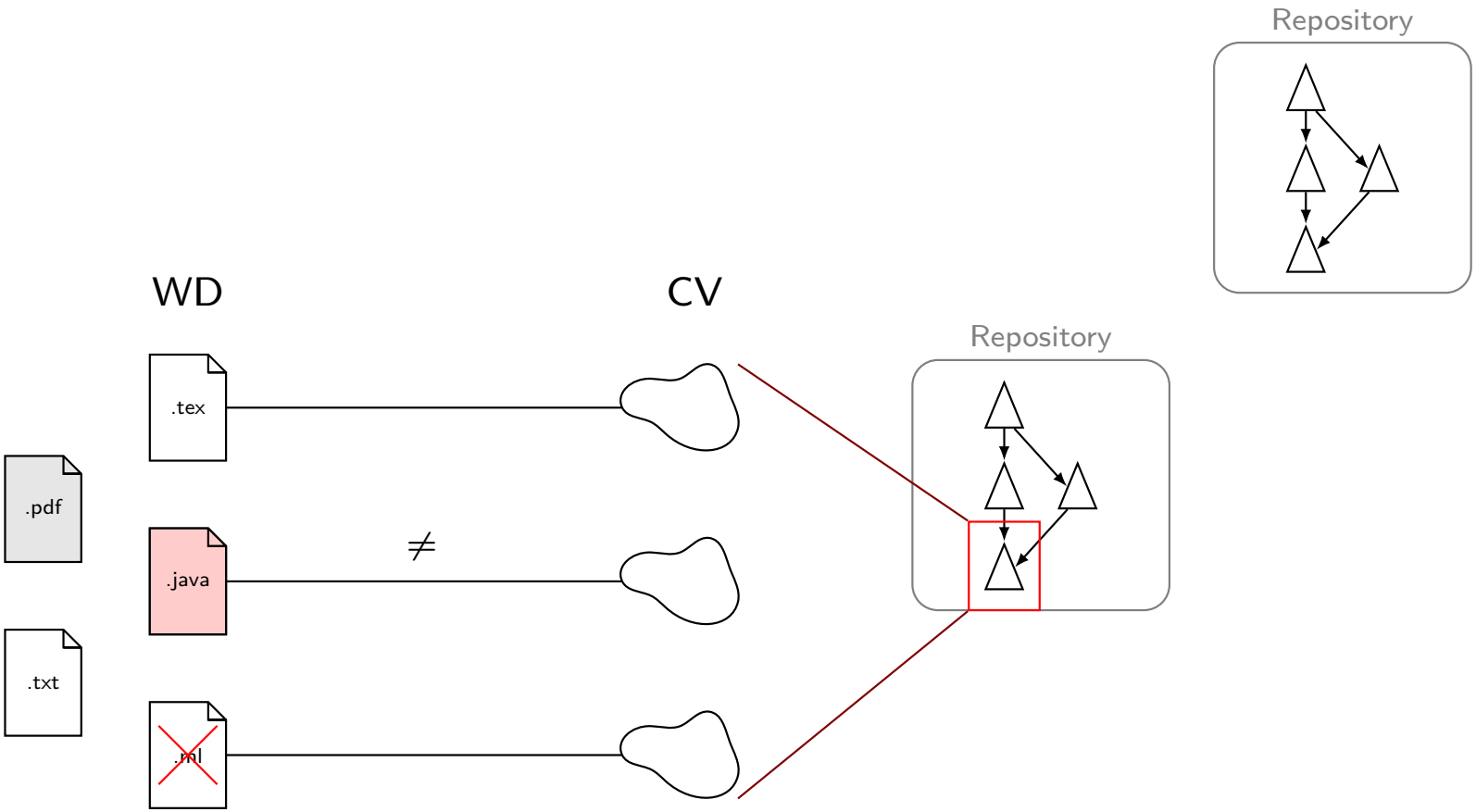


Diving into Git core

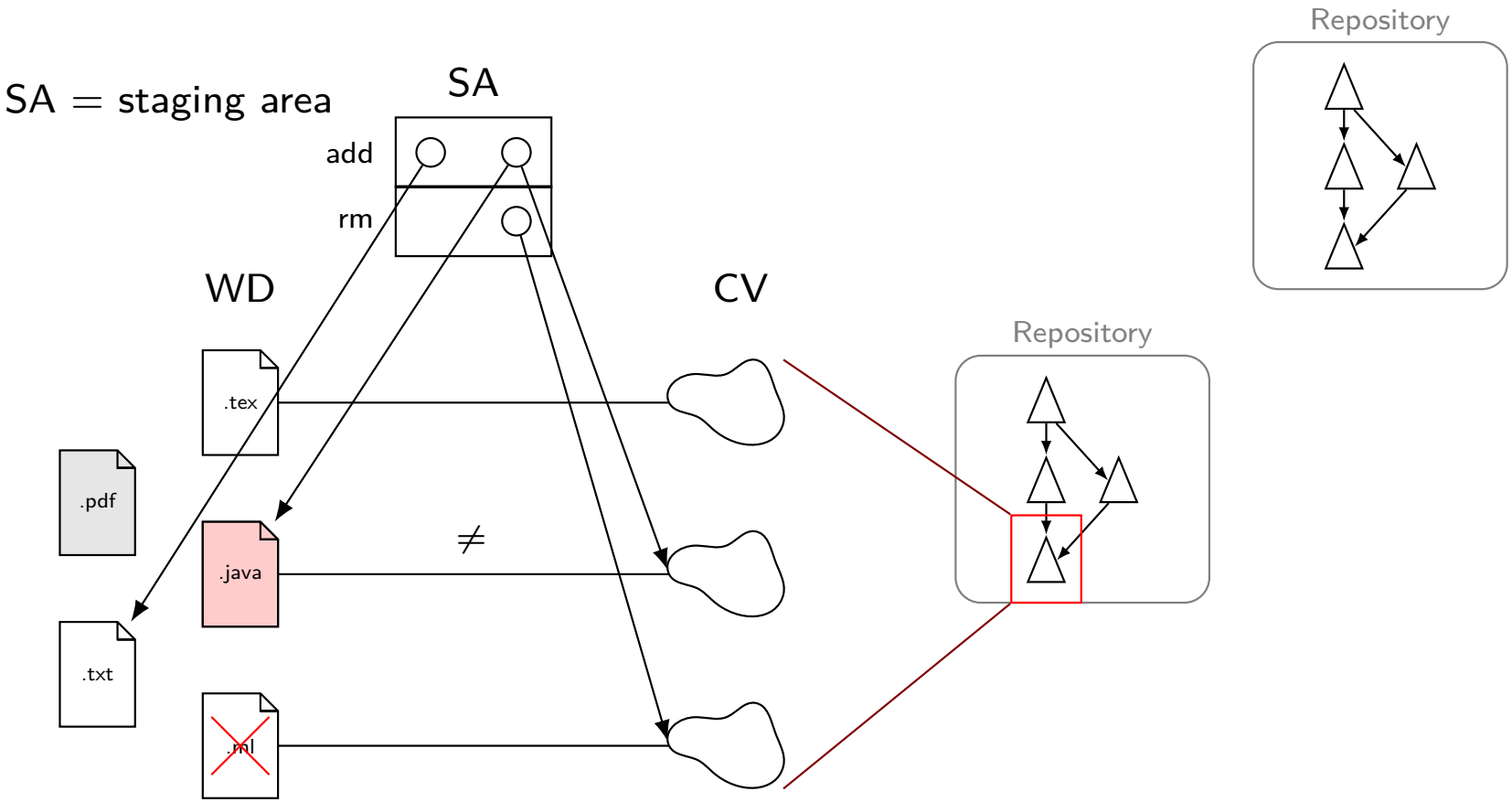
WD = working directory



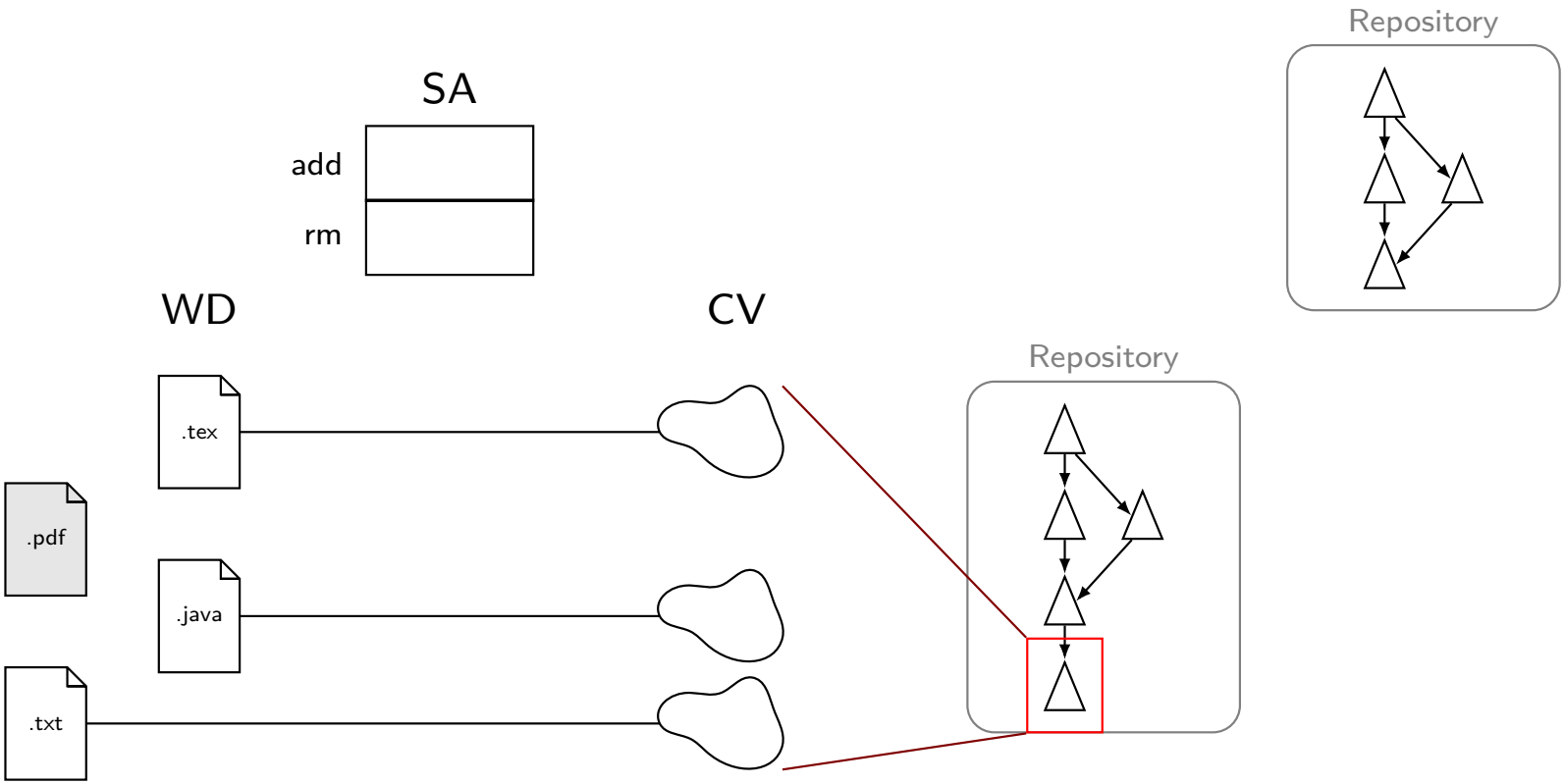
Diving into Git core



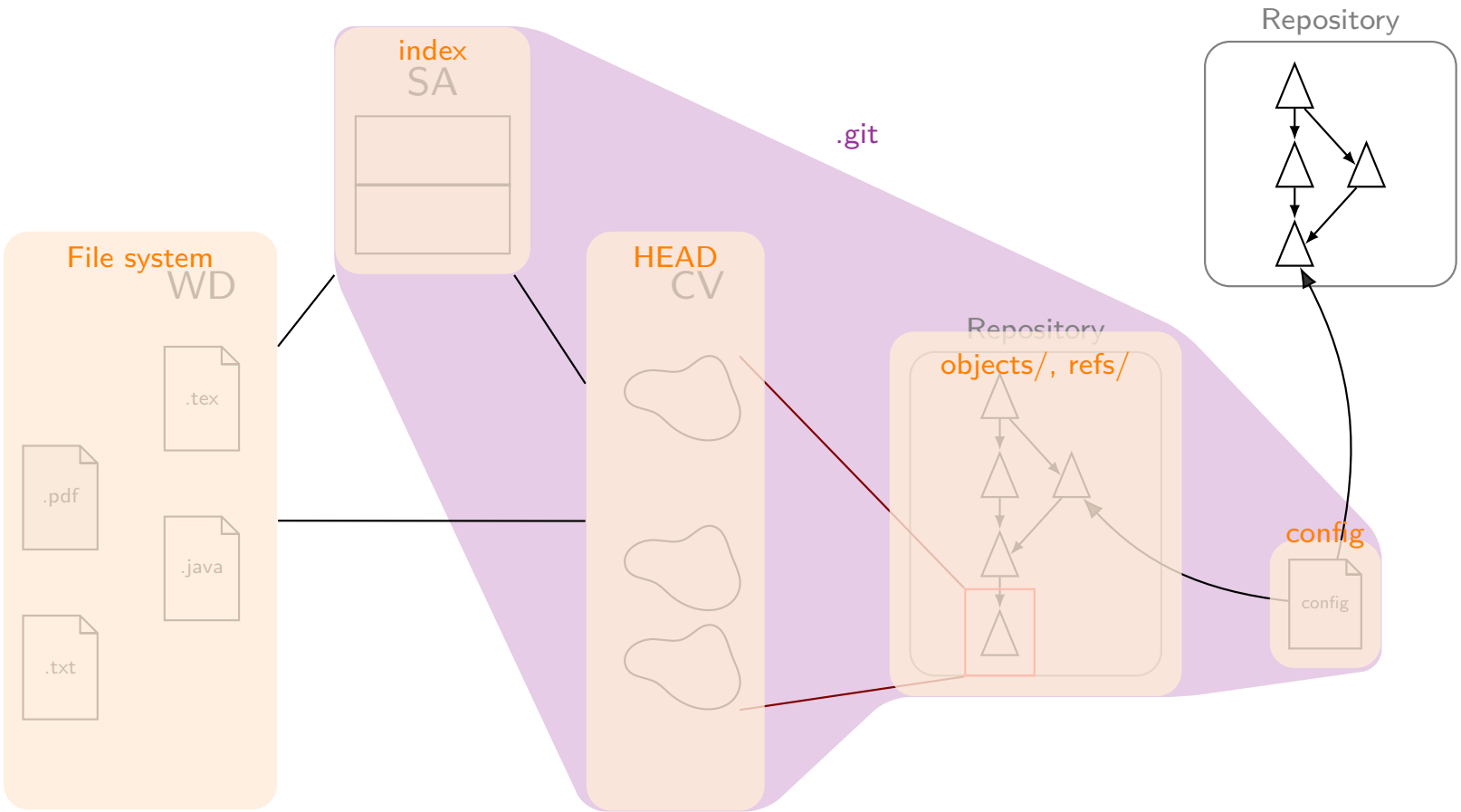
Diving into Git core



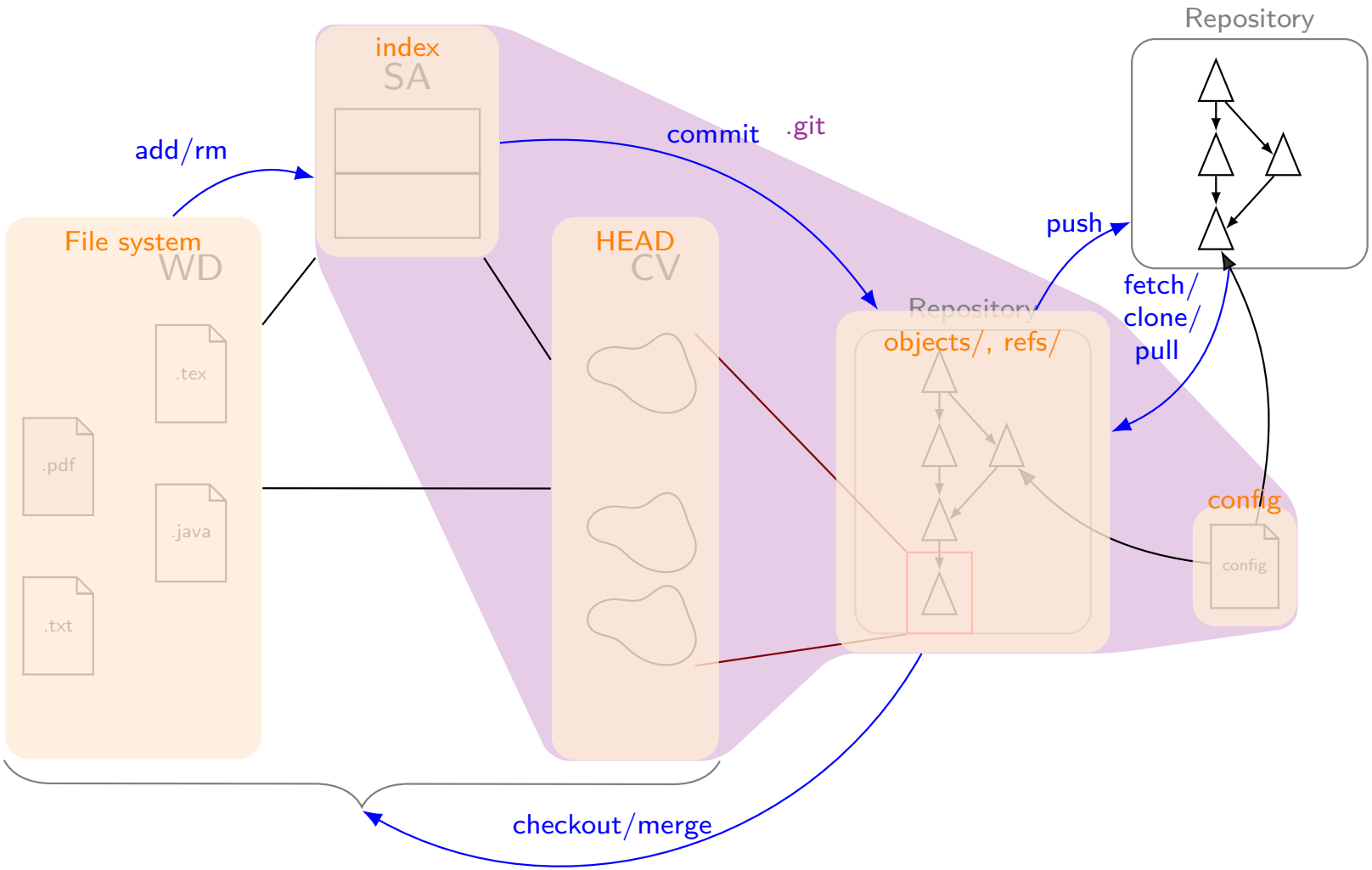
Diving into Git core



Diving into Git core



Diving into Git core



Structure of a versioned project with Git

```
jcbach@minitel2#11:32:20>taf-ilsd$ls -lah
total 72K
drwxr-xr-x  14 jcbach jcbach 4.0K Aug 12 09:58 .
drwxr-xr-x 258 jcbach jcbach  12K Aug 19 10:54 ..
drwxr-xr-x   7 jcbach jcbach 4.0K Jan 10  2025 adm
drwxr-xr-x   6 jcbach jcbach 4.0K Aug 12 09:58 contenu
drwxr-xr-x   2 jcbach jcbach 4.0K Aug 12 09:58 descriptions
drwxr-xr-x   8 jcbach jcbach 4.0K Jun 25  2019 fiches
drwxr-xr-x   2 jcbach jcbach 4.0K May  3  2024 fichesPASS
drwxr-xr-x   7 jcbach jcbach 4.0K Aug 22 10:42 .git
-rw-r--r--   1 jcbach users    50 Dec 11  2024 .gitignore
drwxr-xr-x   2 jcbach users  4.0K Aug 12 09:58 logo
drwxr-xr-x   4 jcbach users  4.0K Aug 12 09:58 poster
drwxr-xr-x   4 jcbach users  4.0K Aug 19 15:07 presentations
drwxr-xr-x   4 jcbach users  4.0K Aug 12 09:58 projets-procom
-rw-r--r--   1 jcbach users   494 Aug 12 09:58 README
drwxr-xr-x   2 jcbach jcbach 4.0K Dec  1  2020 reunions
drwxr-xr-x   6 jcbach users  4.0K Aug 13 17:52 semainerentree
```

.git structure

```
jcbach@minitel2#11:36:28>.git$ls -lah
total 84K
drwxr-xr-x   7 jcbach jcbach 4.0K Aug 22 10:42 .
drwxr-xr-x  14 jcbach jcbach 4.0K Aug 12 09:58 ..
-rw-r--r--   1 jcbach jcbach   20 Sep 17  2024 COMMIT_EDITMSG
-rw-r--r--   1 jcbach users   276 Dec 11  2024 config
-rw-r--r--   1 jcbach jcbach   73 Jun 25  2019 description
-rw-r--r--   1 jcbach jcbach  107 Aug 19 14:59 FETCH_HEAD
-rw-r--r--   1 jcbach users    21 Dec 11  2024 HEAD
drwxr-xr-x   2 jcbach jcbach 4.0K Jun 25  2019 hooks
-rw-r--r--   1 jcbach users   27K Aug 22 10:42 index
drwxr-xr-x   2 jcbach jcbach 4.0K Jun 25  2019 info
drwxr-xr-x   3 jcbach jcbach 4.0K Jun 25  2019 logs
drwxr-xr-x 260 jcbach jcbach 4.0K Aug 19 14:59 objects
-rw-r--r--   1 jcbach users    41 Aug 19 14:59 ORIG_HEAD
-rw-r--r--   1 jcbach jcbach  114 Jun 25  2019 packed-refs
drwxr-xr-x   5 jcbach jcbach 4.0K Sep 23  2020 refs
```

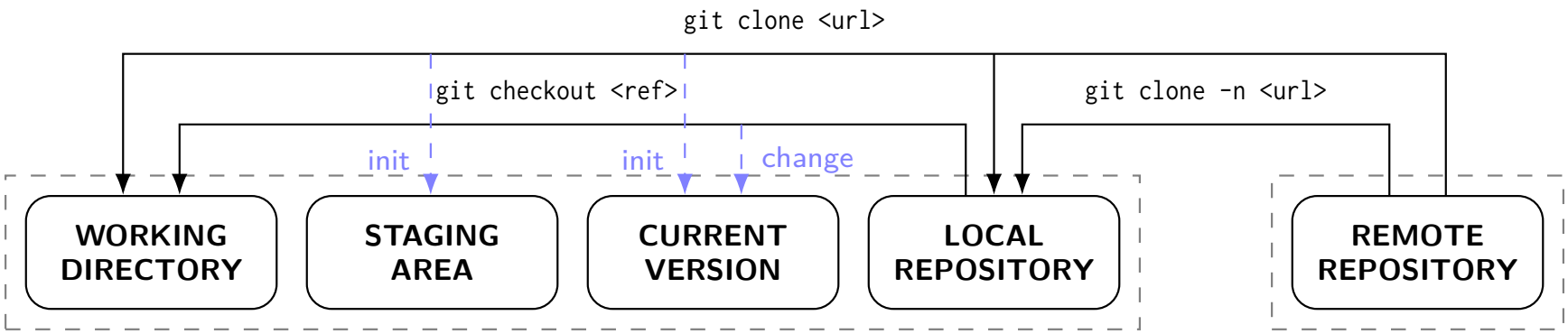
Progress

Git by the example

- Practical use cases in order to learn few commands
  - setting up a new repository (`init`, `remote url`)
  - retrieving a repository (`clone`)
  - making changes in the working repository (`status`)
  - updating the remote environment (`add`, `commit`, `push`)
  - checking differences after changes (`diff`)
  - updating dev environment (`fetch`, `pull`)
  - diverging/branching (`branch`, `merge`, `checkout`)
  - ...
- Non-exhaustive use cases
- Workflows

Let's have a look at the terminal!
(I'll probably forget the slides)

Retrieving a repository



`$> git clone -n <url>`

only creates the `.git` directory

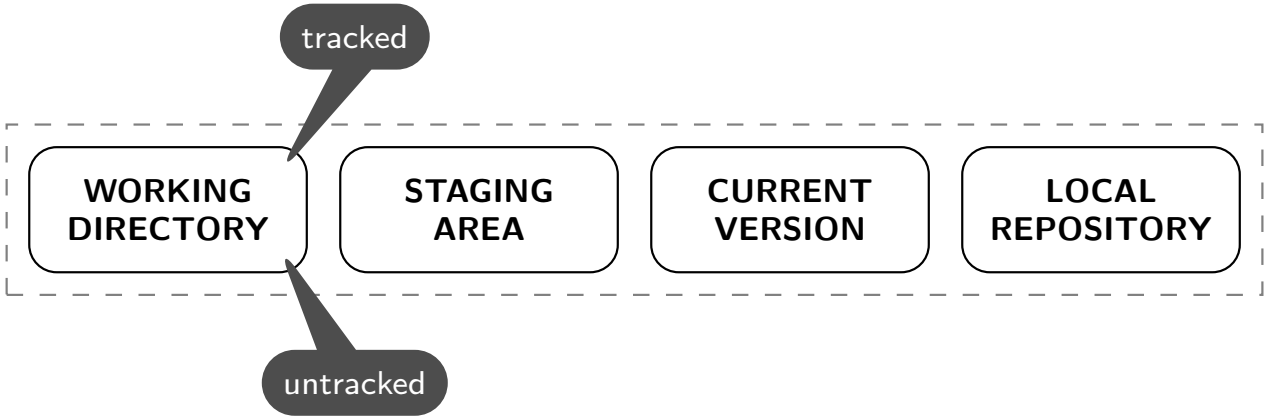
`$> git checkout <ref>`

retrieves files from local repository into the working directory

`$> git clone <url>`

creates the `.git` directory and retrieves files into the working directory; `clone = clone -n + checkout`

Making changes in the working directory

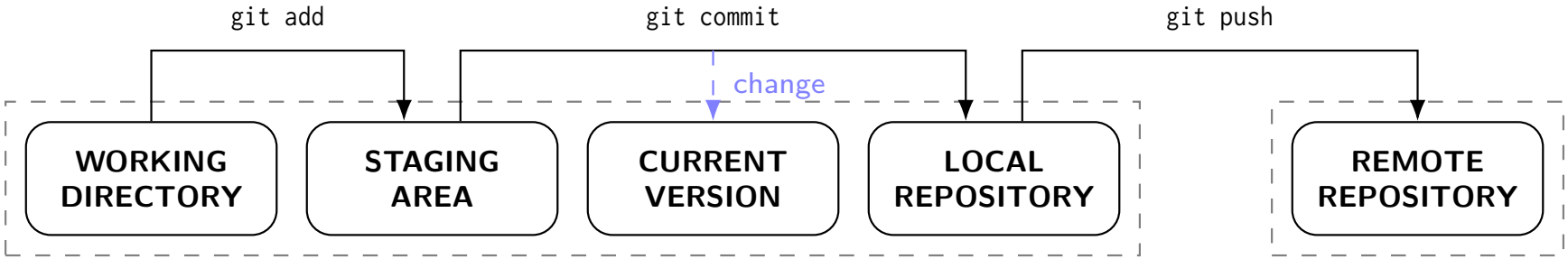


Checking the current state

```
$> git status
```

```
On branch main
Your branch is up to date with 'origin/main'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   file1
        ...
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4
        ...
no changes added to commit (use "git add" and/or "git commit -a")
```

Updating the remote environment



Example

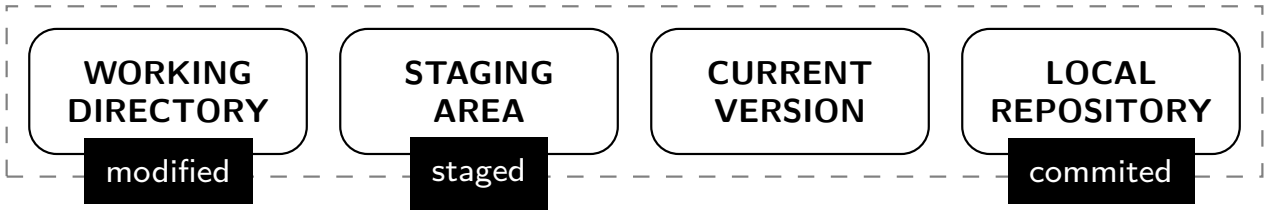
```
$> git add file1 file2 file3 ...
```
add in the index of the staging area

```
$> git commit -m ``add my super new feature''
```

```
...
```

```
$> git push
```
push into the remote repository

Checking differences after changes



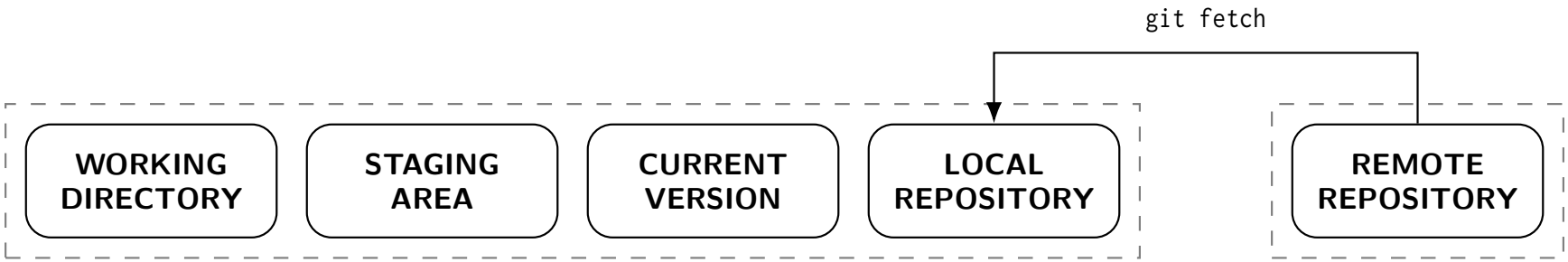
Diff commands

`$> git diff`

`$> git diff --staged`

`$> man git-diff` will help you

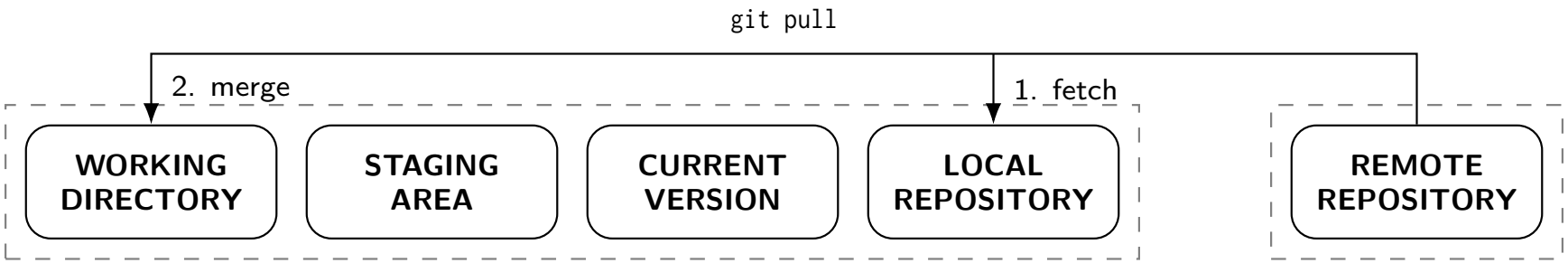
Updating dev environment (`fetch`)



`$> git fetch`

- retrieves updates from the remote repository
- is safe
  - does not affect working directory ⇒ cannot lose uncommited changes,
  - no automated merge

Updating dev environment (pull)



`$> git pull`

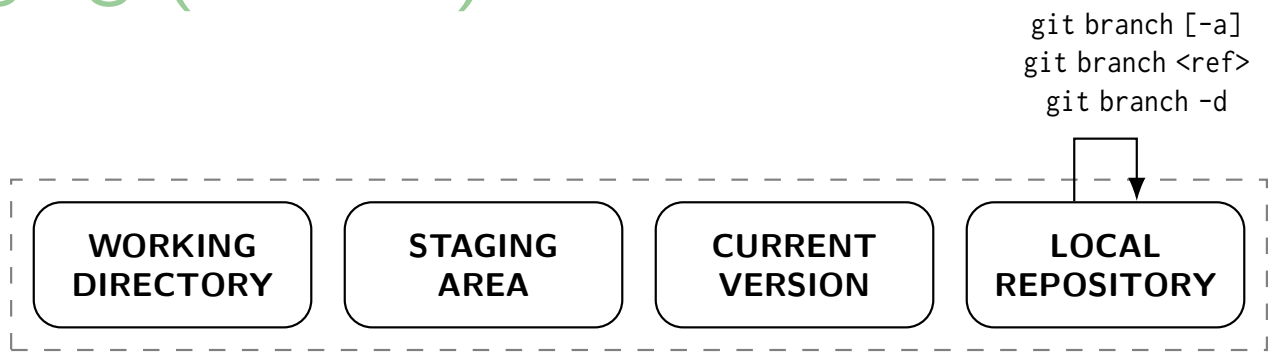
retrieves updates from the remote repository and merge them with the working directory

- `git merge`: to be seen few slides later

Diverging: vocabulary

- a branch = a reference to a version
  - can be seen as a “local checkpoint” (another says like a bookmark)
- branching
  - creating a named reference to a version
  - the common way to work without messing with the main line

Diverging (branch)



`$> git branch`

list local branches

`$> git branch -a`

list all (local and remote) branches

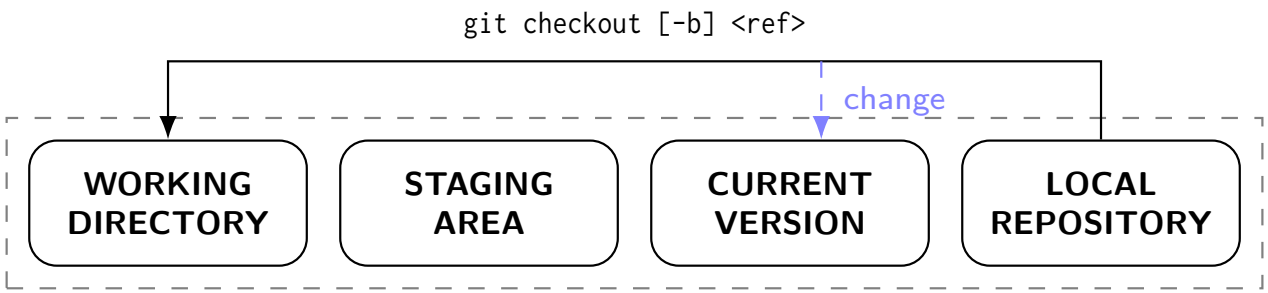
`$> git branch <ref>`

creates a named branch from the current branch

`$> git branch -d <ref>`

deletes a named branch

Diverging (checkout)



`$> git checkout <ref>`

changes the current branch

`$> git checkout -b <ref>`

creates a branch from the current branch and changes to it (= `git branch` + `git checkout`)

Diverging (merge)

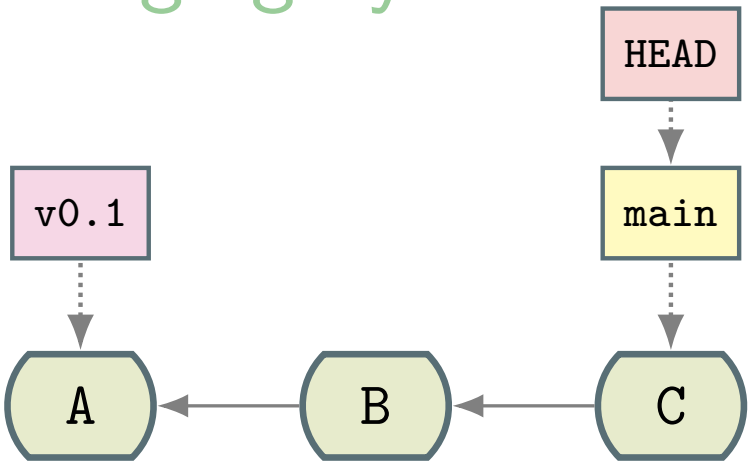
- Starting point: 2 branches (`main` + `newawesomefeature`), `HEAD` points to `main`

```
$> git merge newawesomefeature
```

integrate changes from `newawesomefeature` branch into `main`

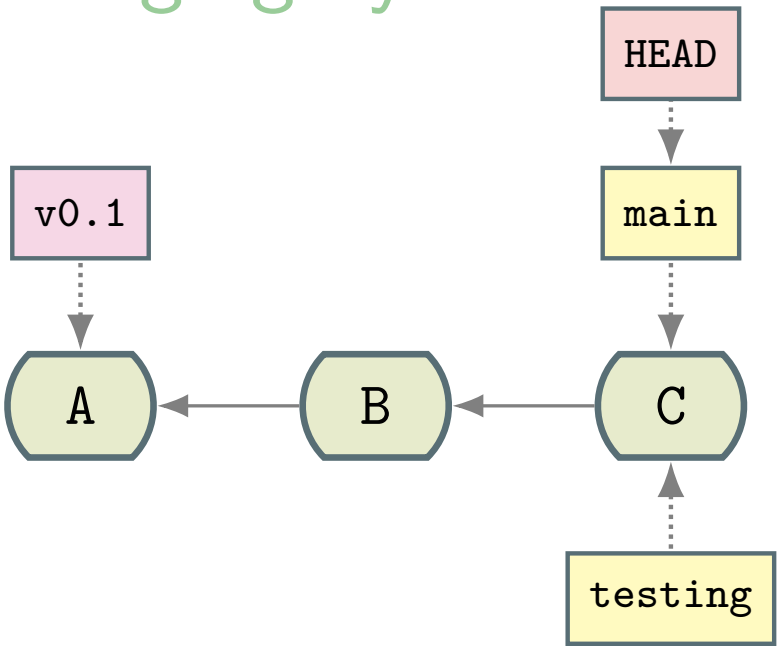
- Two situations
  - no conflict: changes from `newawesomefeature` are integrated in the main (local) line, time to push
  - conflicts: resolution needed in order to be able to push
- Conflict resolution:
  1. fix the conflicts (edit the files, keep/remove stuff)
  2. add the changes
  3. commit

Diverging by the example



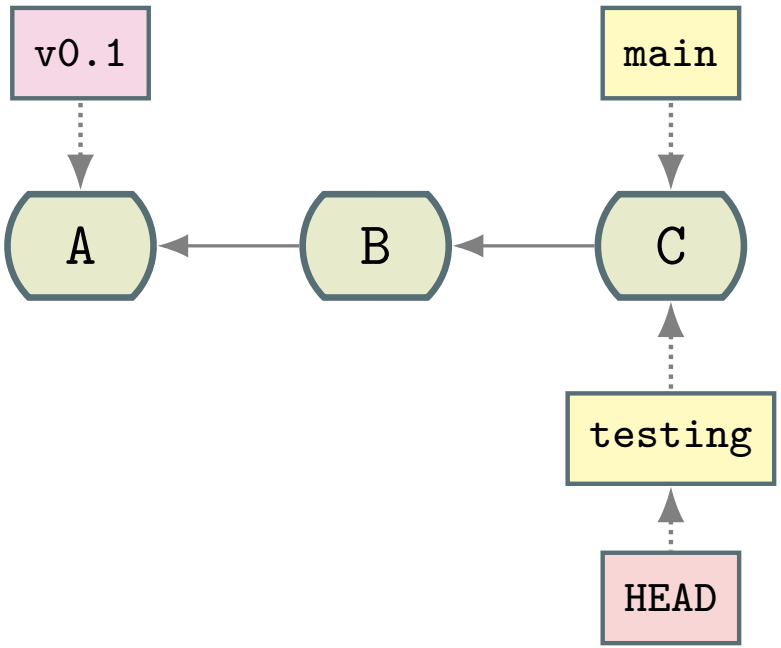
Initial situation

Diverging by the example



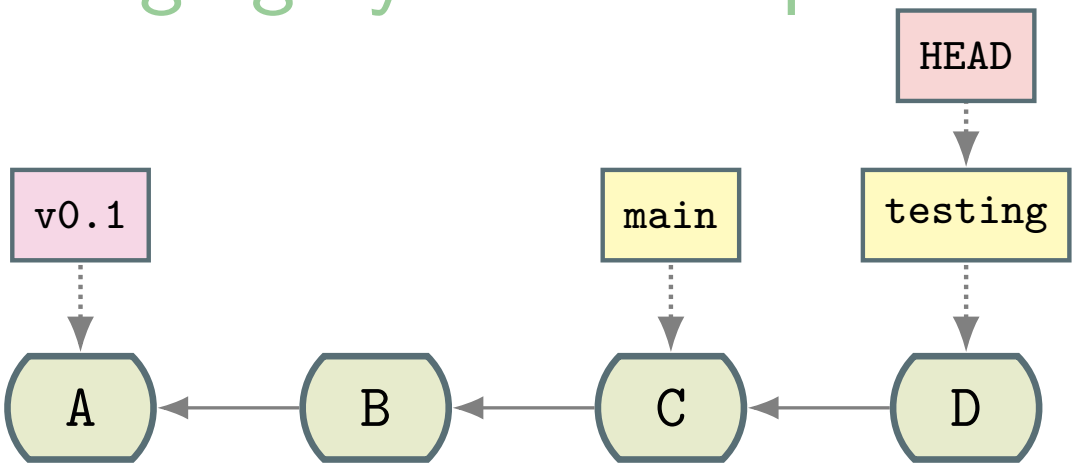

```
$> git branch testing
```

Diverging by the example



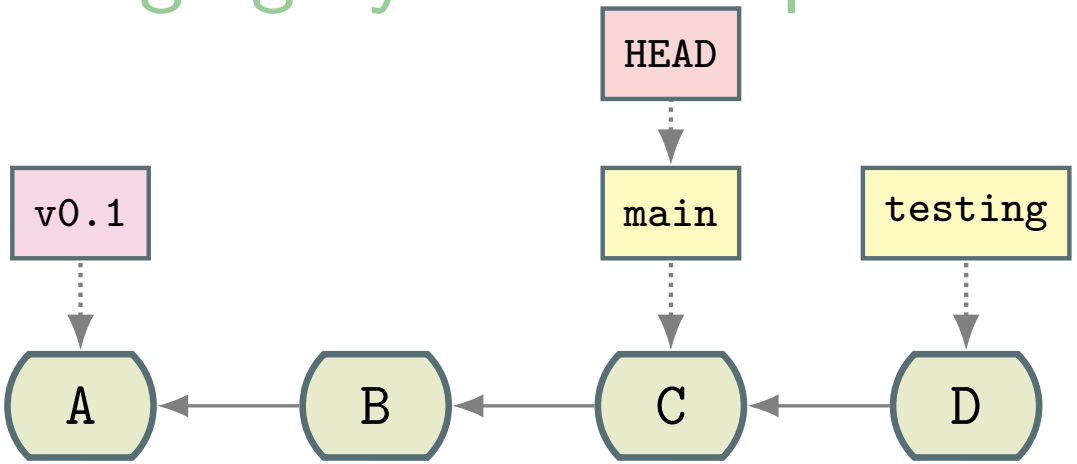

```
$> git checkout testing
```

Diverging by the example



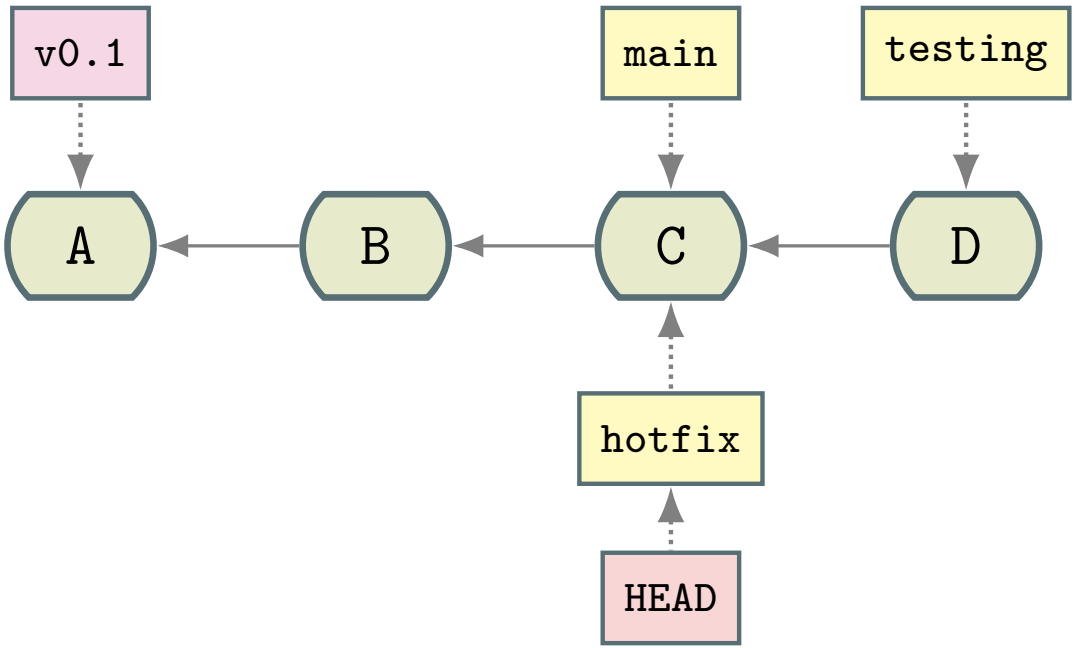
One commit later

Diverging by the example



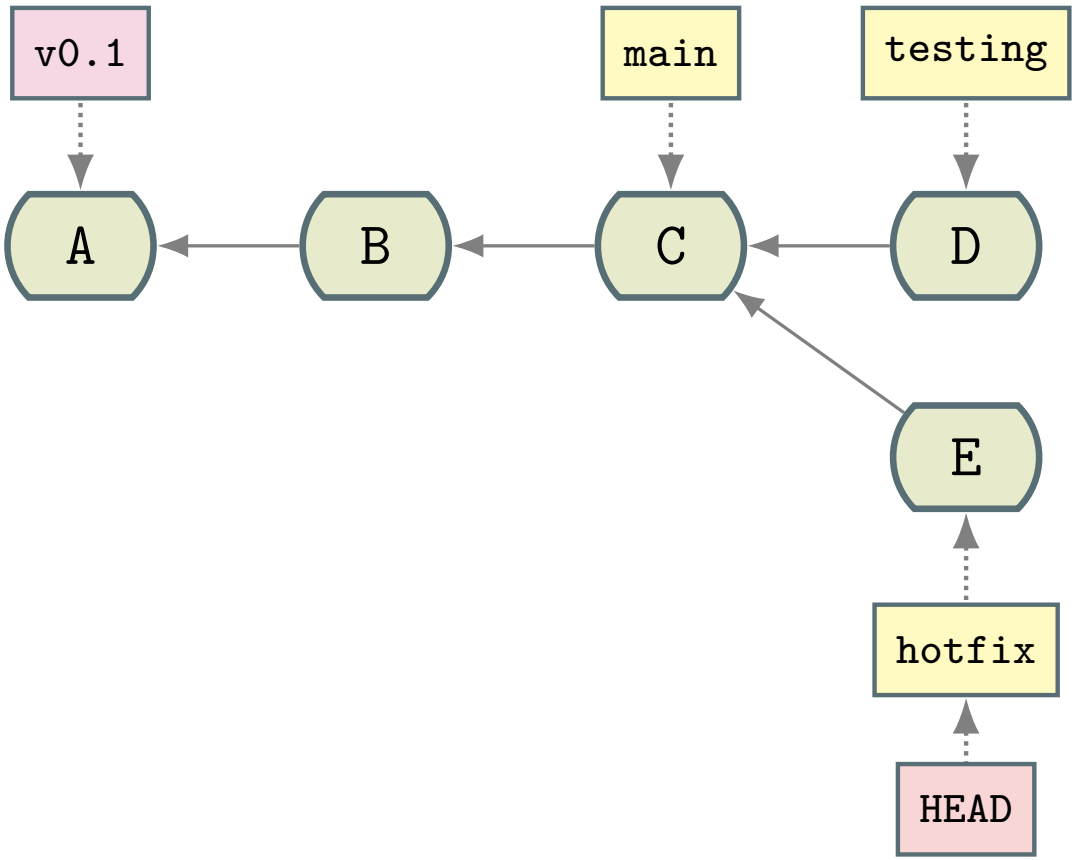

```
$> git checkout main
```

Diverging by the example



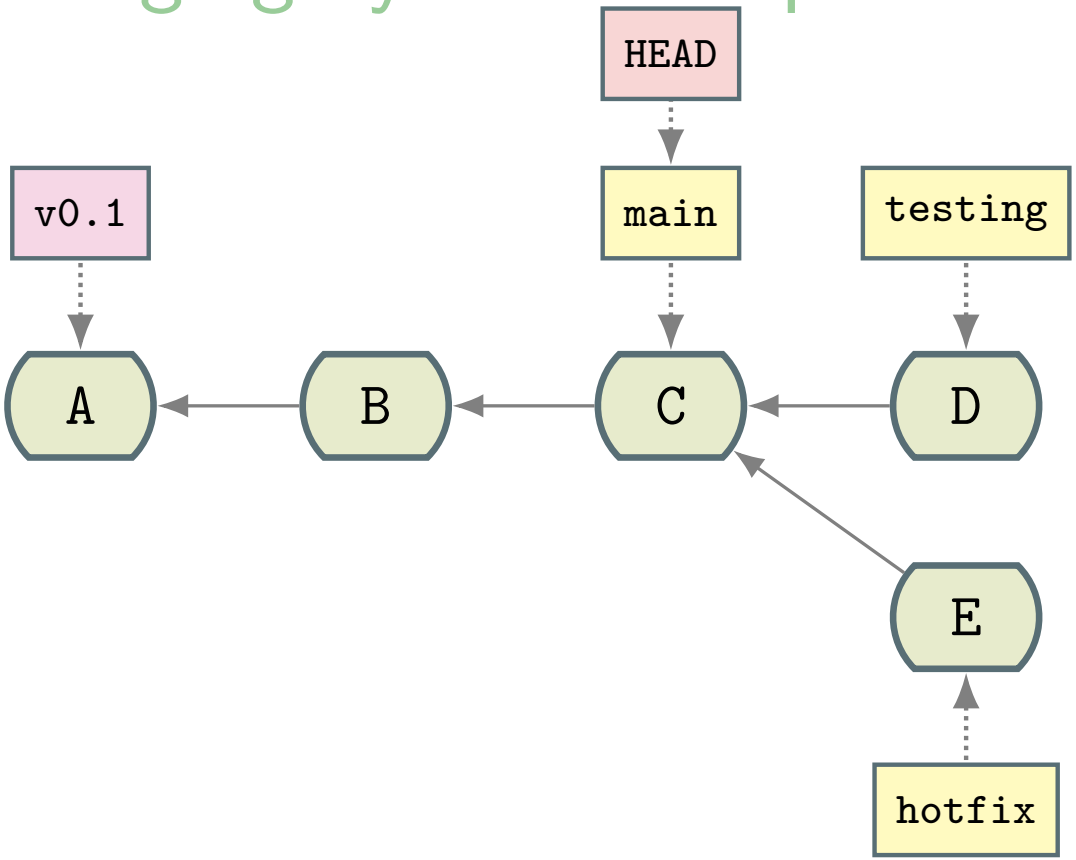

```
$> git checkout -b hotfix
```

Diverging by the example



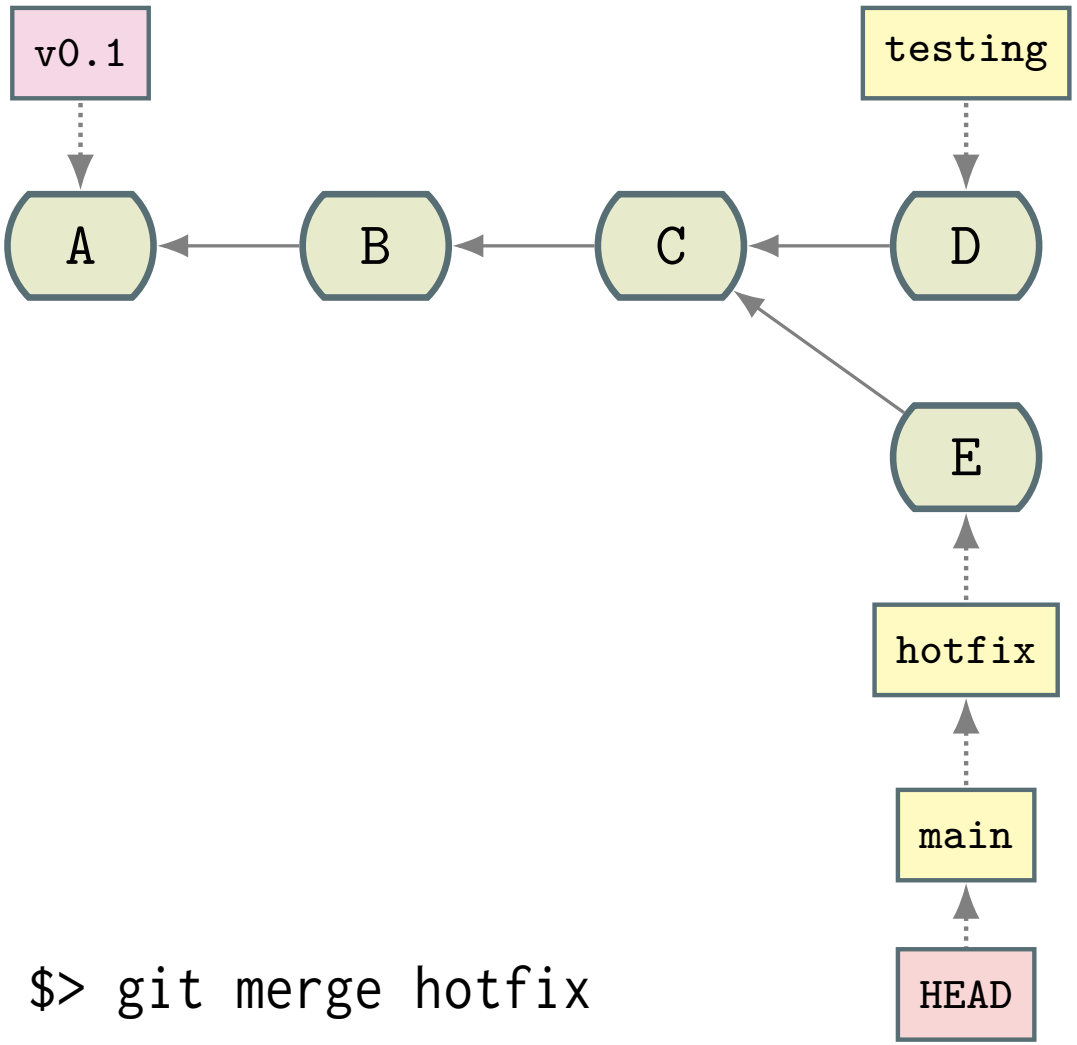
One commit later

Diverging by the example



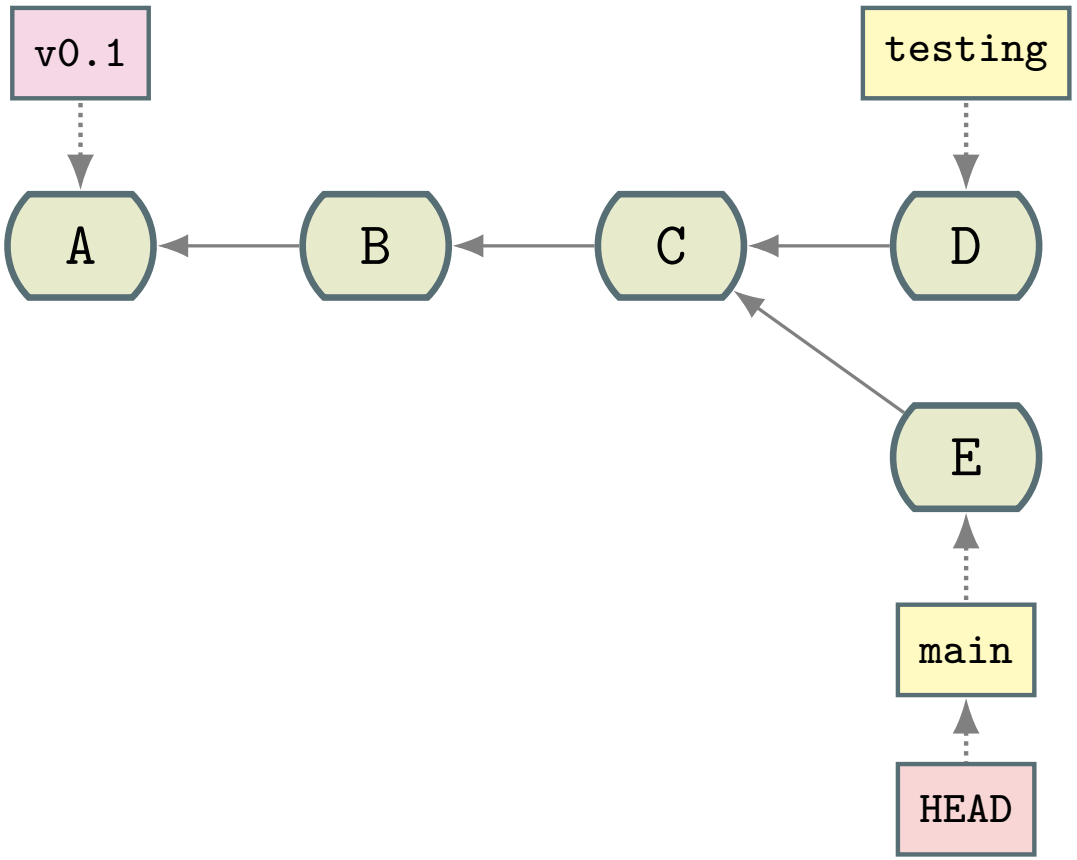

```
$> git checkout main
```

Diverging by the example



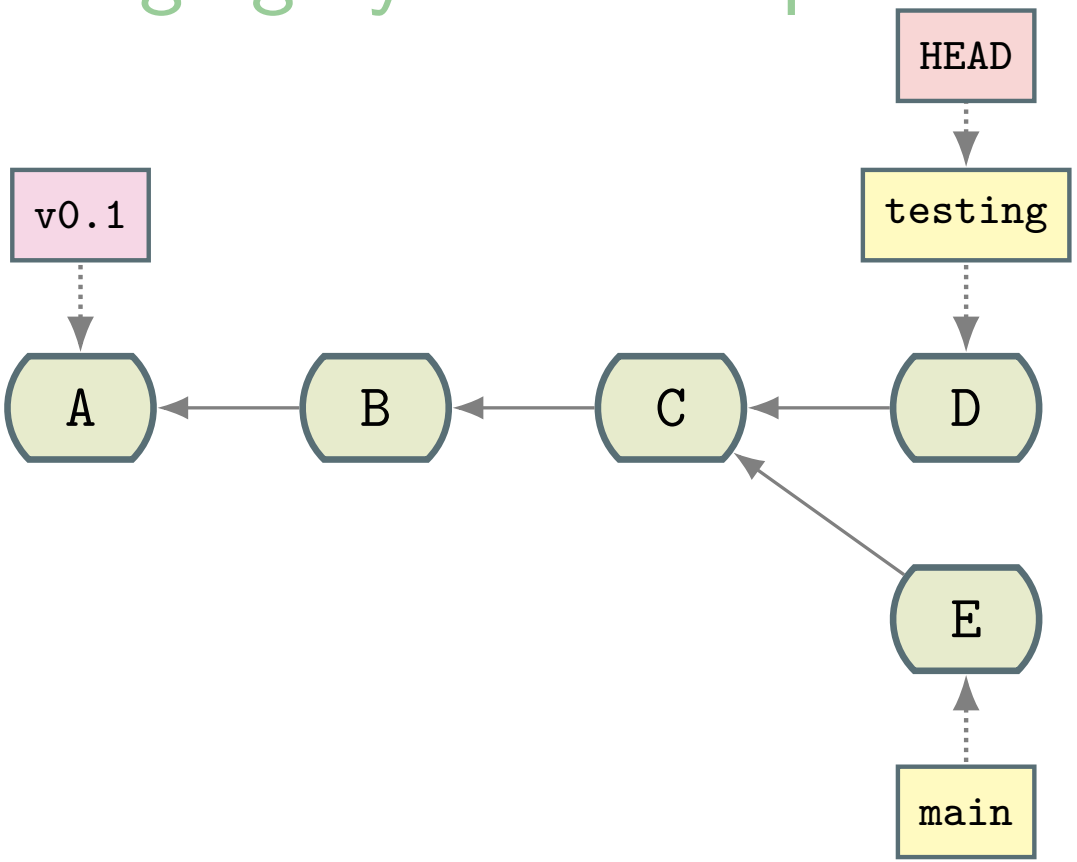

```
$> git merge hotfix
```

Diverging by the example



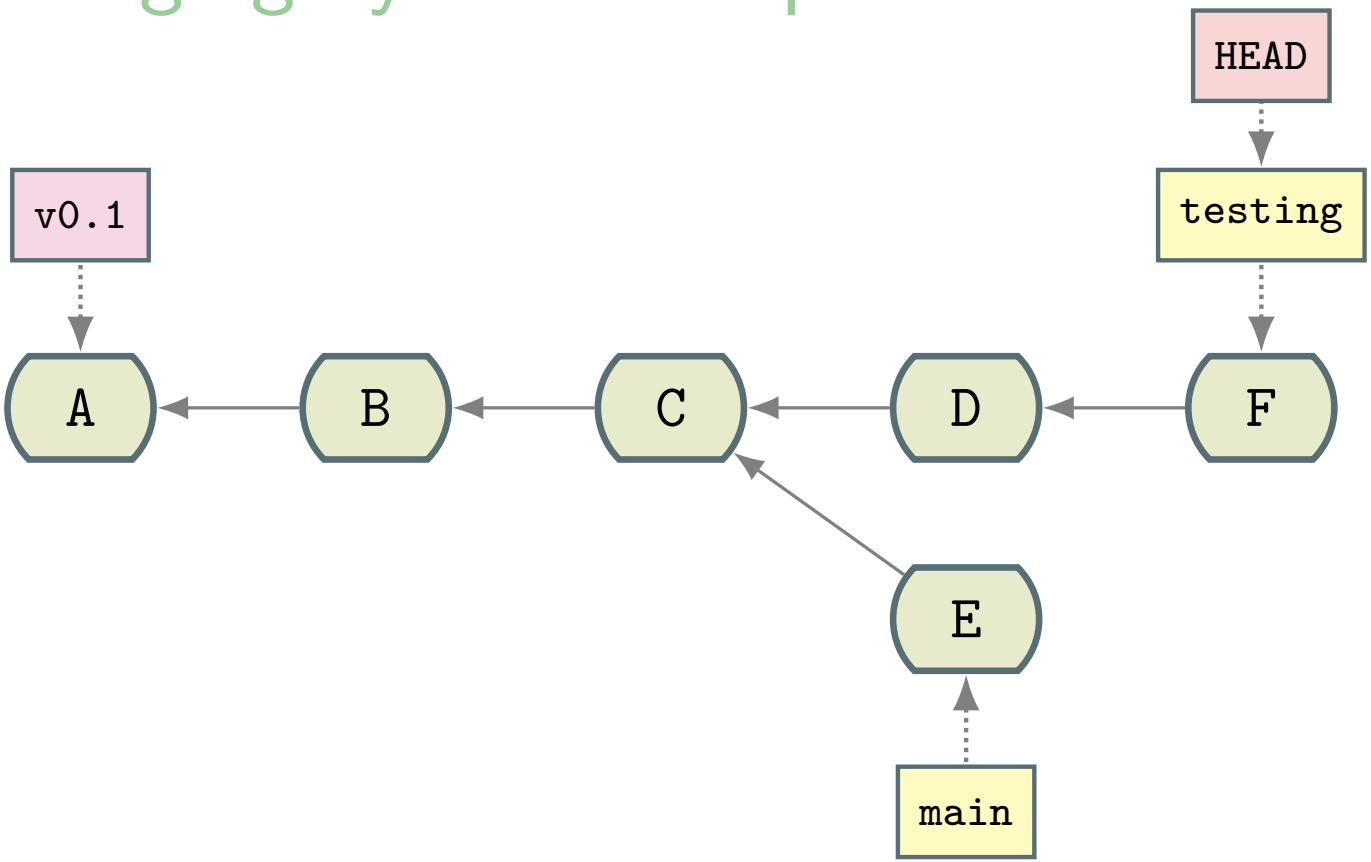

```
$> git branch -d hotfix
```

Diverging by the example



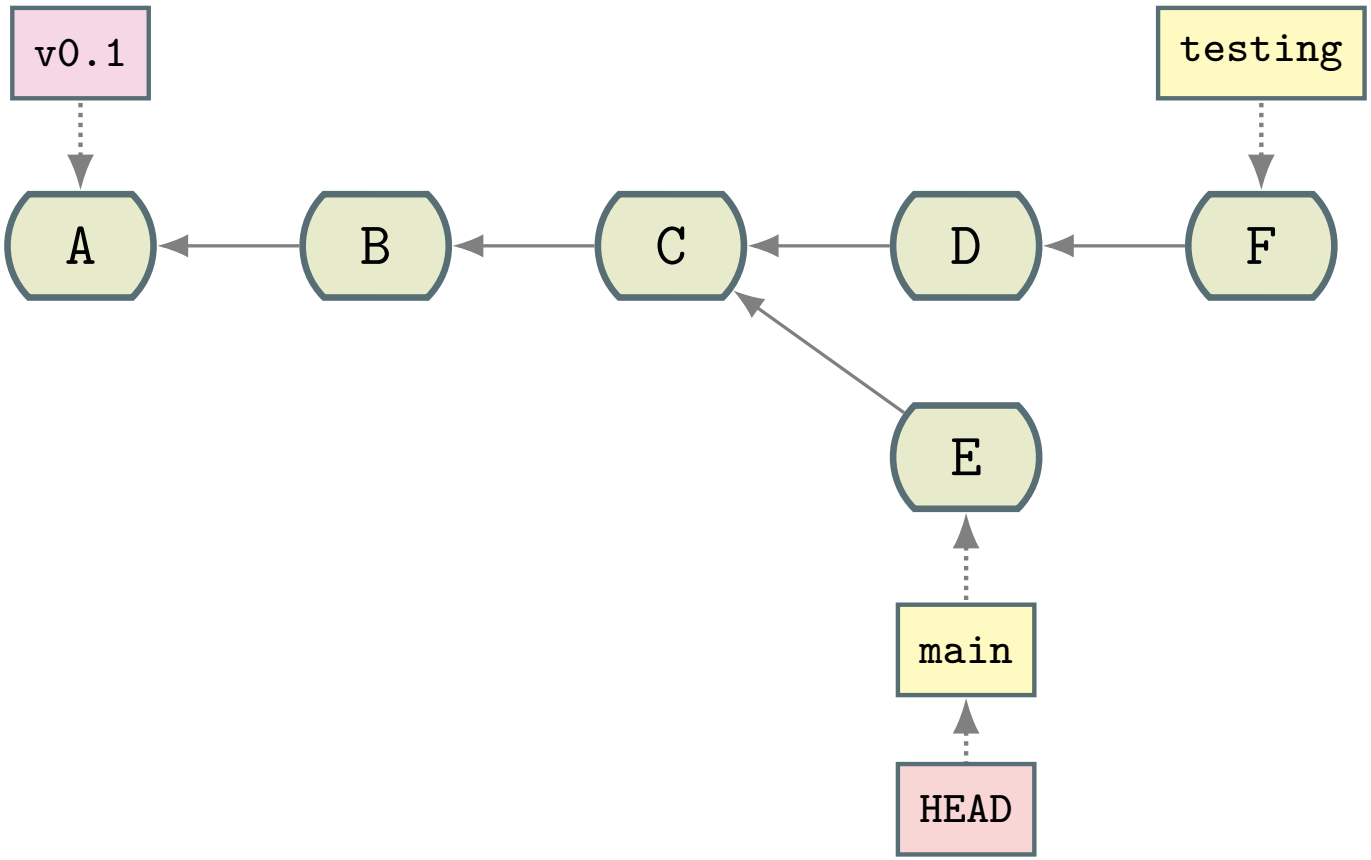

```
$> git checkout testing
```

Diverging by the example



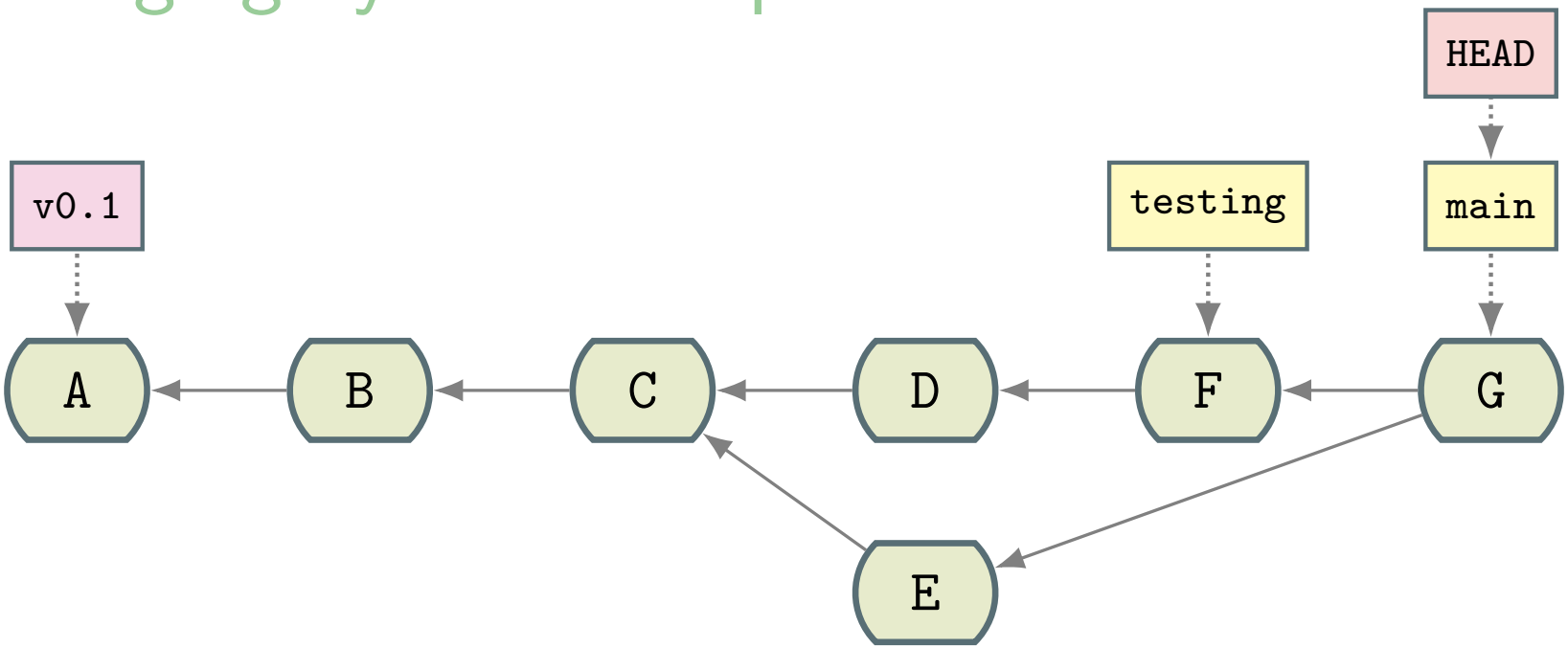
One commit later

Diverging by the example



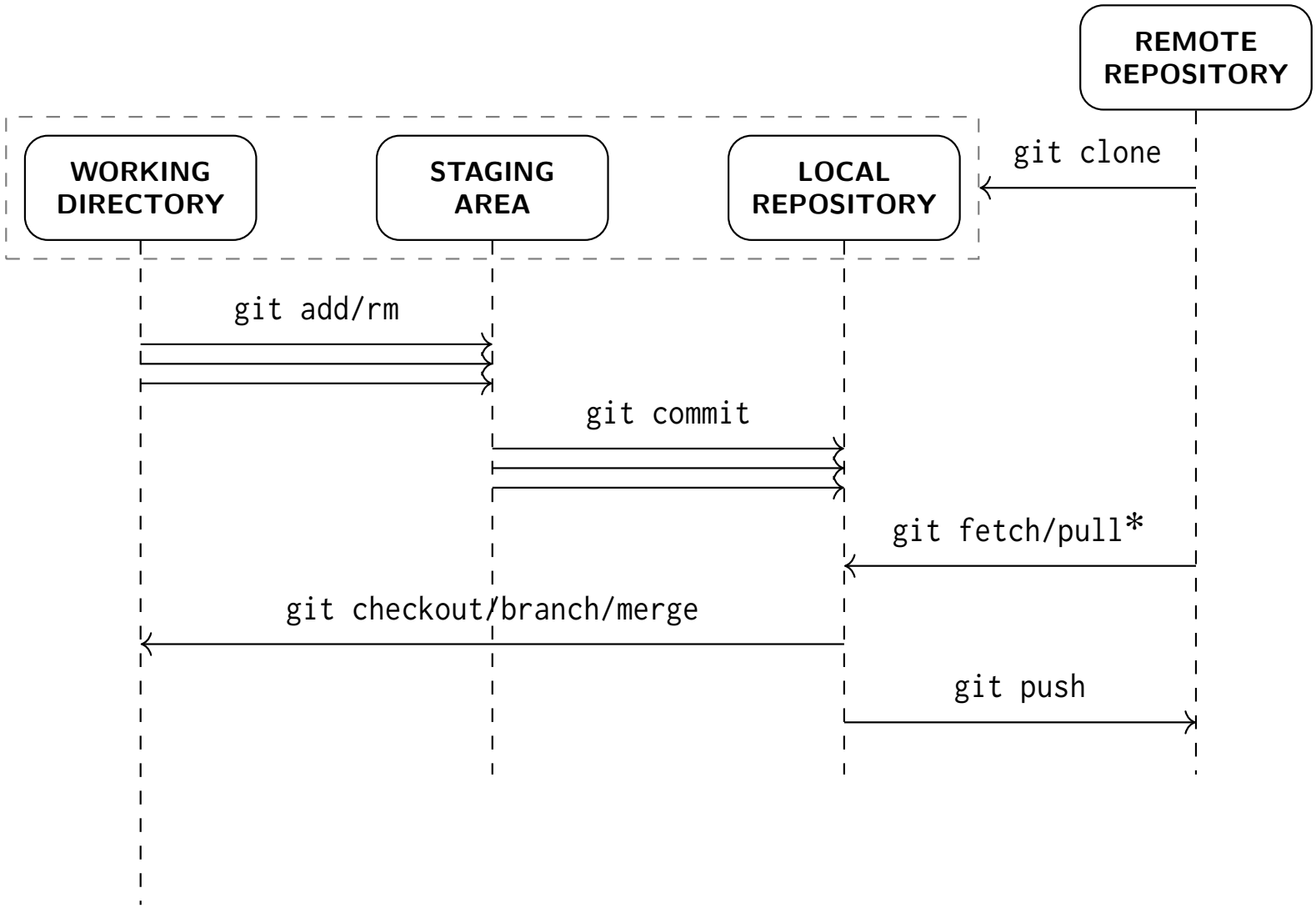

```
$> git checkout main
```

Diverging by the example




```
$> git merge testing
```

Summary of a typical Git workflow



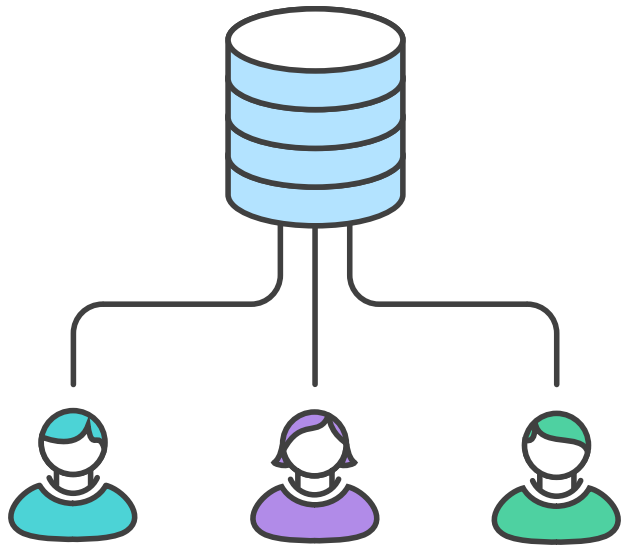
Progress

Adopting a workflow

- One tool, many usages
- Tools alone do not solve development problems
- Need of a process that fits the team
- Many possible Git workflows (examples later)
  - centralised workflow
  - feature branch workflow
  - gitflow workflow
  - forking workflow
  - ...

Centralised workflow

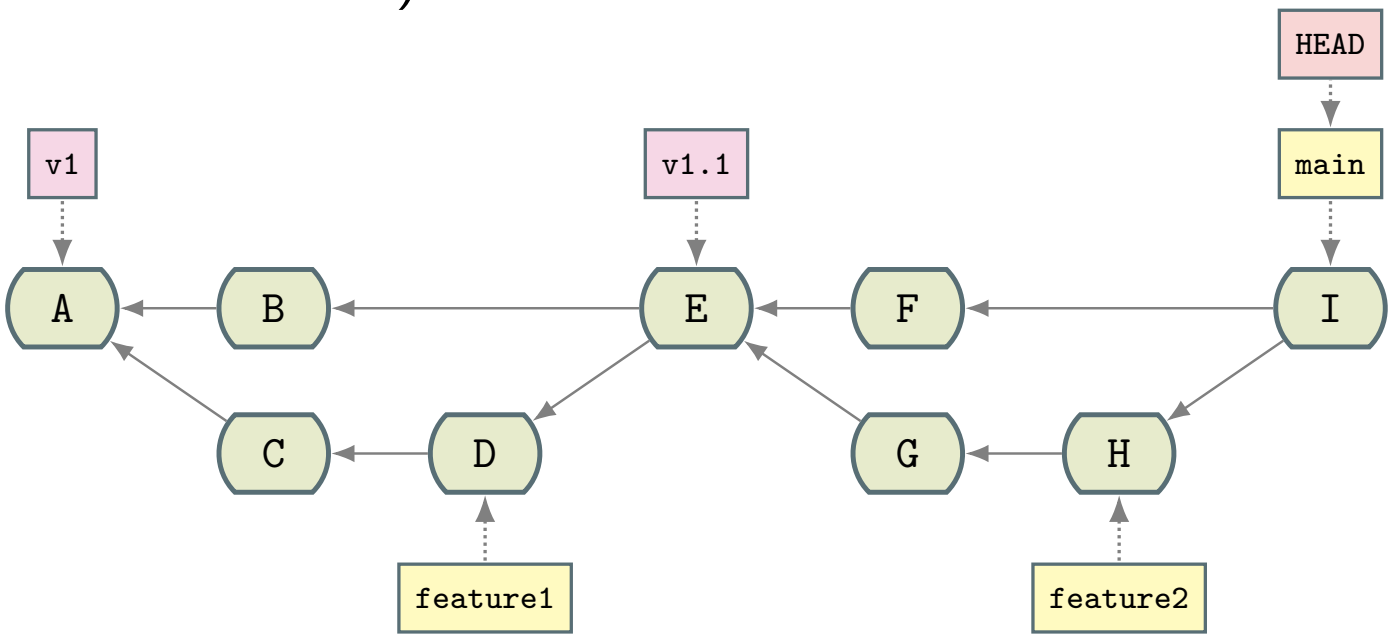
- One central repository, one branch (main)
- Common when coming from centralised systems like Subversion
- Common for small size teams
- Easy to understand for a newcomer



Source: Atlassian

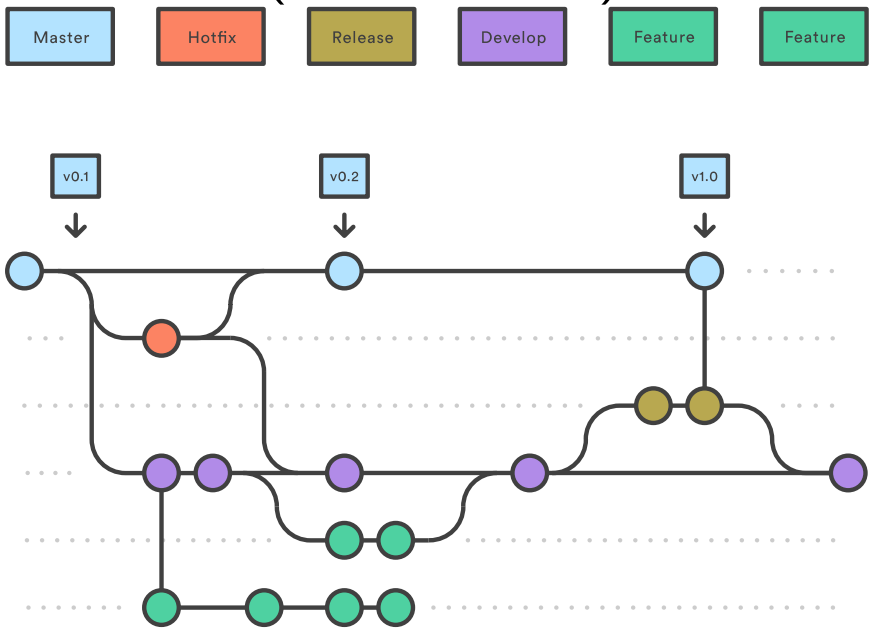
Feature branch workflow

- Central repository + main branch = official project history
- One branch per feature: no direct commit on the main branch
- Feature branches are pushed to the central repository
- Branches are then merged (after pull requests, feedbacks, conflict resolutions)



Gitflow workflow

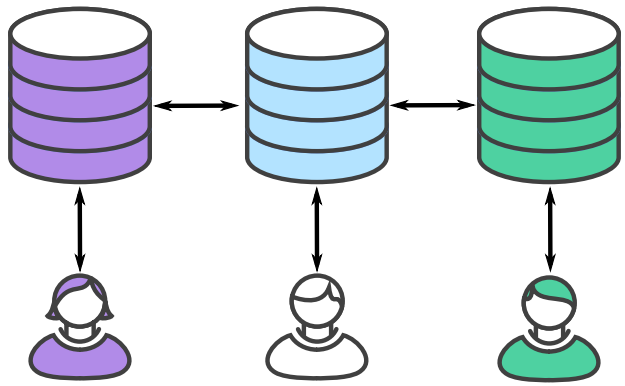
- Strict branching model designed around the project release
- Well-suited for large projects with deadlines (releases)
- One branch one role, workflow defines their interactions
- Can be combined with feature branch workflow
- Project history = main (the releases) + development branch



Source: Atlassian

Forking workflow

- One serverside repository per developer
- Each developer manages her repository and make pull requests to the reference repository
- Typical model when contributing to a FLOSS project hosted on GitHub: “Fork us on GitHub”



Source: Atlassian

Workflows: summary

- Chosen workflow depends on the team's concerns and organisation
  - no one-size-fits-all Git workflow
- Feature workflow: business domain oriented
- Forking and gitflow workflows: repository oriented
- What is a good workflow?
  - enhance or limit team efficiency?
  - scale with team size?
  - easy to undo mistakes and errors?
  - impose any new unnecessary cognitive overhead to the team?
  - does it limit conflicts?

Progress

VCS pratices

- Git
  - useful and powerful tool
  - but a tool alone does not solve all problems. It can also create ones

  $\Rightarrow$ developers do not only need tools, but also working processes
- Good practices
  - formalizing the process/workflow
  - coordinating with co-workers
  - testing before sending changes
  - updating before sending a change
  - commiting meaningful changes
  - commiting often
  - adding meaningful messages for commits
  - not commiting generated files
  - short-lived branches

VCS in practice: how to start?

- By practicing
  - during every lab sessions, even in non-CS context
  - at home
- One usually needs a server to host repositories
  - but it is not mandatory: you can use Git in serverless mode!
- Some questions to ask before chosing
  - do you want to make your project public?
  - is there any security, privacy or IP problems with the project?
  - is your project a cornerstone of your business?
- Your answers should drive your choices of VCS hosting
  - simple and free non-professional account on an open platform
  - paid service on a platform
  - installation of your own VCS server

Git in practice: which platform to start with?

- IMTA infrastructure for academic projects and for learning:
  - Gitlab: `https://gitlab-df.imt-atlantique.fr/`
- Many platforms can be used without any fee:
  - Assembla: `https://www.assembla.com/`
  - Bitbucket: `https://bitbucket.org/`
  - Codeberg: `ihttps://codeberg.org/`
  - Gitea: `https://about.gitea.com/`
  - Gitlab: `https://about.gitlab.com/`
  - GitHub: `https://github.com/`
  - Sourcehut: `https://sourcehut.org/`

  ... and probably many other
- You can also install your own server!

Serverless mode: a simple way to start with Git

- Git can also be used without any other host

1. `$> mkdir mycode`
2. `$> cd mycode`
3. `$> git init`
   initialize a new Git repository

- That type of Git repository can be shared
  - as every folder (copy/paste on an USB key,)
  - or using a Git command to add a remote repository (it has to exist)

`$> git remote add <name> <url>`

Git integration

- If you use a mainstream IDE, Git is probably already integrated
  - Eclipse
    - Window > Perspective > Open Perspective > Other > Git
  - vscode
    - Ctrl+Shift+G
    - nice and useful plugins: *Git graph* and *GitLens*
  - IntelliJ
    - Alt+‘
  - well-configured Vim or Emacs: you don’t need any help 😎
    - maybe a TUI: *tig*, *lazygit*

Conclusion

- Basic principles of VCS
  - basic principles
  - two main families: centralised vs decentralised
  - tools diversity
- Some good practices for VCS usage
- Importance of a workflow
  - should be simple
  - should enhance the team productivity
  - should be oriented by business requirements
- VCS usage should be an habit, not a constraint
- Basics for a specific (but probably the most common) VCS: Git
  - discover Git in the practical work!

Resources

- VCS
  - https://homes.cs.washington.edu/~mernst/advice/version-control.html
  - https://betterexplained.com/articles/a-visual-guide-to-version-control/
  - https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/
- Git
  - https://git-scm.com/
  - https://git-scm.com/book/en/v2/ (Pro Git book)
  - http://justinhileman.info/article/git-pretty/
  - https://betterexplained.com/articles/aha-moments-when-learning-git/
  - https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html
- Subversion: http://svnbook.red-bean.com/
- Mercurial: https://www.mercurial-scm.org/

Gentle reminder

⚠ Important notes

- if you do not understand something, please ask your questions. *We cannot answer the questions you do not ask*
- if you disagree with us, please say it (politely)
- people don't learn computer science by only reading few academic slides: practicing is fundamental