

# Git: (Distributed) Version Control

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

## Lecture 2

# The Need for Version Control

- Track evolution of a software artifact
  - Development is often non-linear
    - Older versions need to be supported
    - Newer versions need to be developed
  - Development is non-monotonic
    - May need to undo some work, go back to an older version, or track down when a mistake was introduced
- Facilitate team-based development
  - Multiple developers working on a common code base
  - How can project be edited simultaneously?

# Key Idea: A Repository

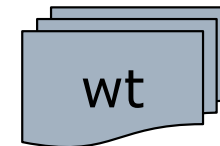
- *Repository* = working tree + store + index
  - Warning: “Repo” often used (incorrectly) to mean just the store or just the working tree
- *Working tree* = project itself
  - Ordinary directory with files & subdirectories
- *Store* = history of project
  - Hidden directory: don't touch!
- *Index* = virtual snapshot
  - Gateway for moving changes in the working tree into the store (aka stage, cache)
- *History* = DAG of *commits*
  - Each node in graph corresponds to a complete snapshot of the entire project

# File Structure of a Repository

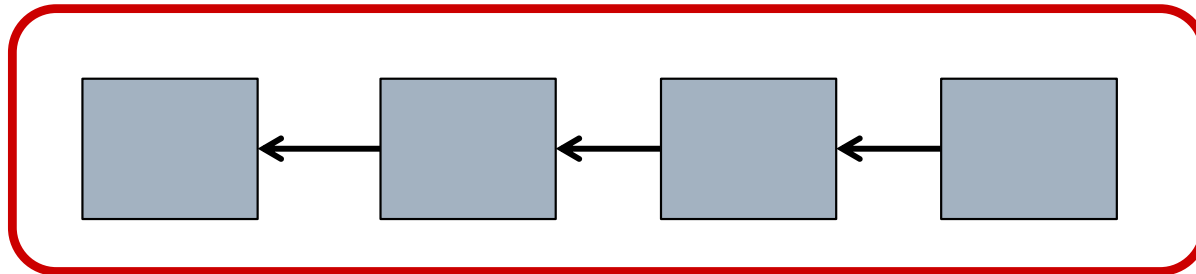
~/mashup/

```
|— css/
|   |— buckeye-alert-resp.css
|   |— demo.css
|— demo-js.html
|— Gemfile
|— Gemfile.lock
|— .git/
|   |— HEAD
|   |— index
|   |— ...etc...
|— .gitignore
|— Rakefile
|— README.md
|— ...etc...
```

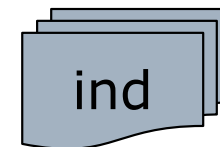
# Conceptual Structure



working tree  
~/mashup/

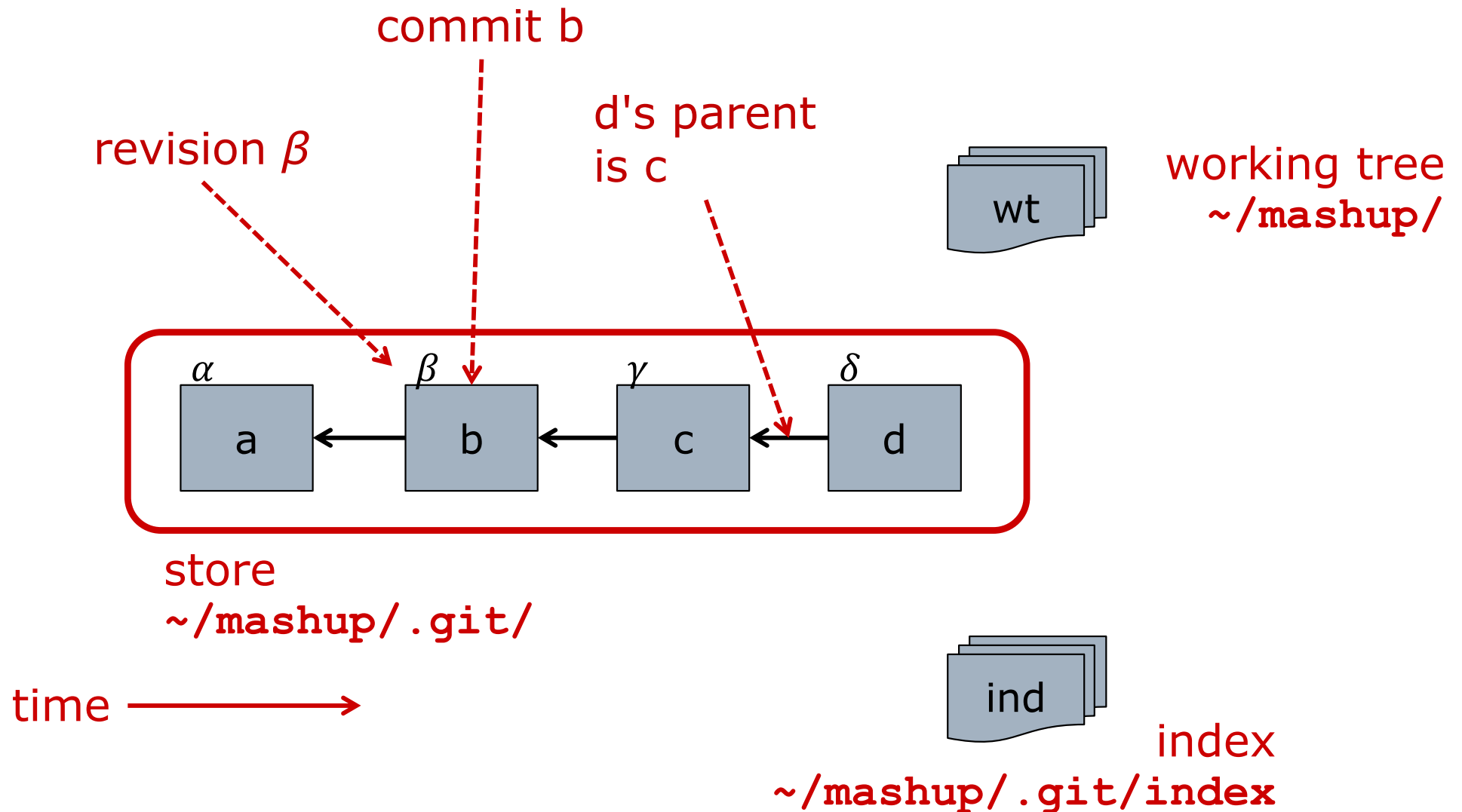


store  
~/mashup/.git/



index  
~/mashup/.git/index

# A History of Commits

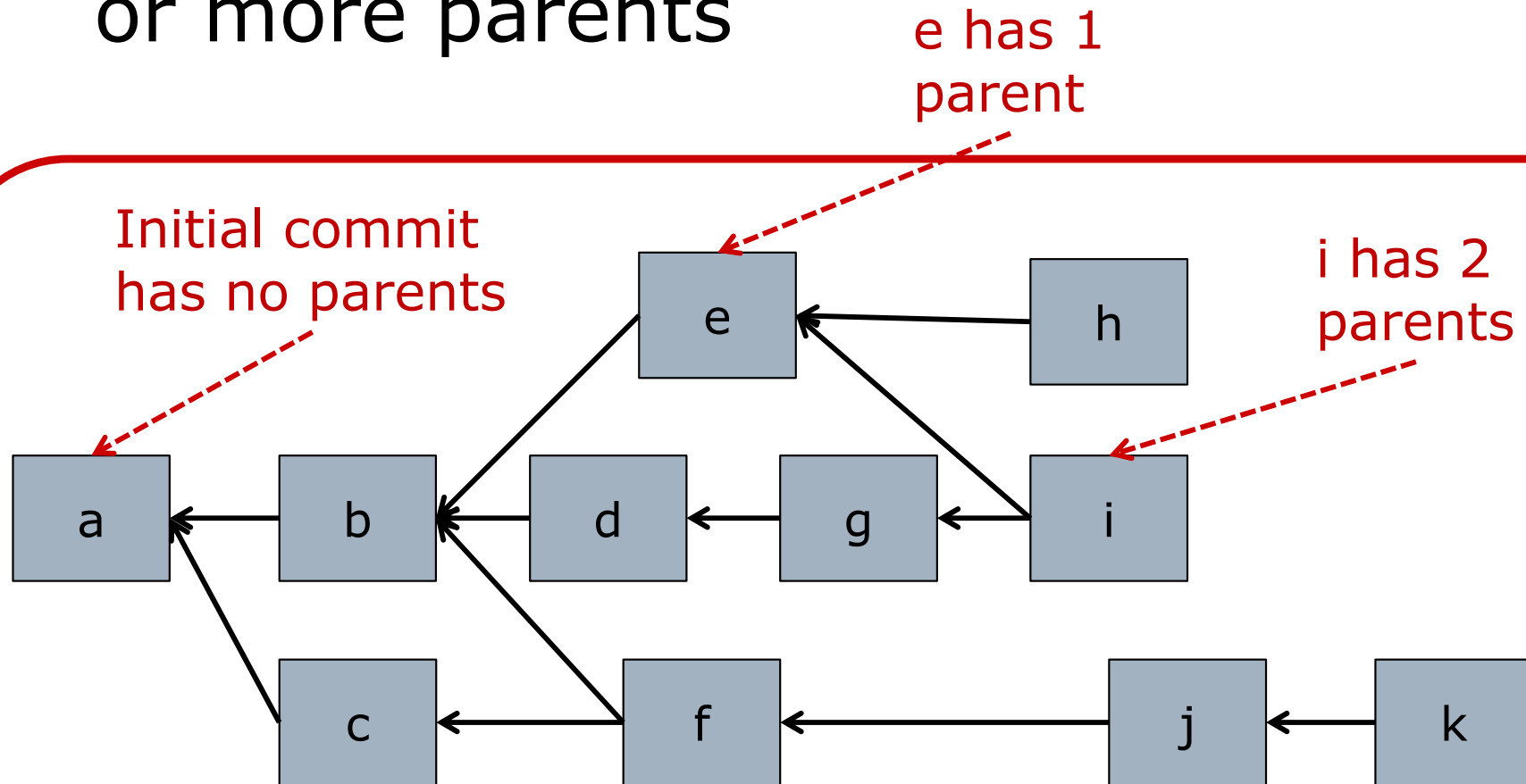


# Commit: Snapshot or Delta

- Each commit represents *both*
  1. A complete snapshot of the project at that point in time (a revision), and
  2. A delta of the changes made (a patch); that is, a diff between snapshots
- Different git actions use different views of a commit
  - View as a snapshot is common
  - Both are useful

# History is a DAG

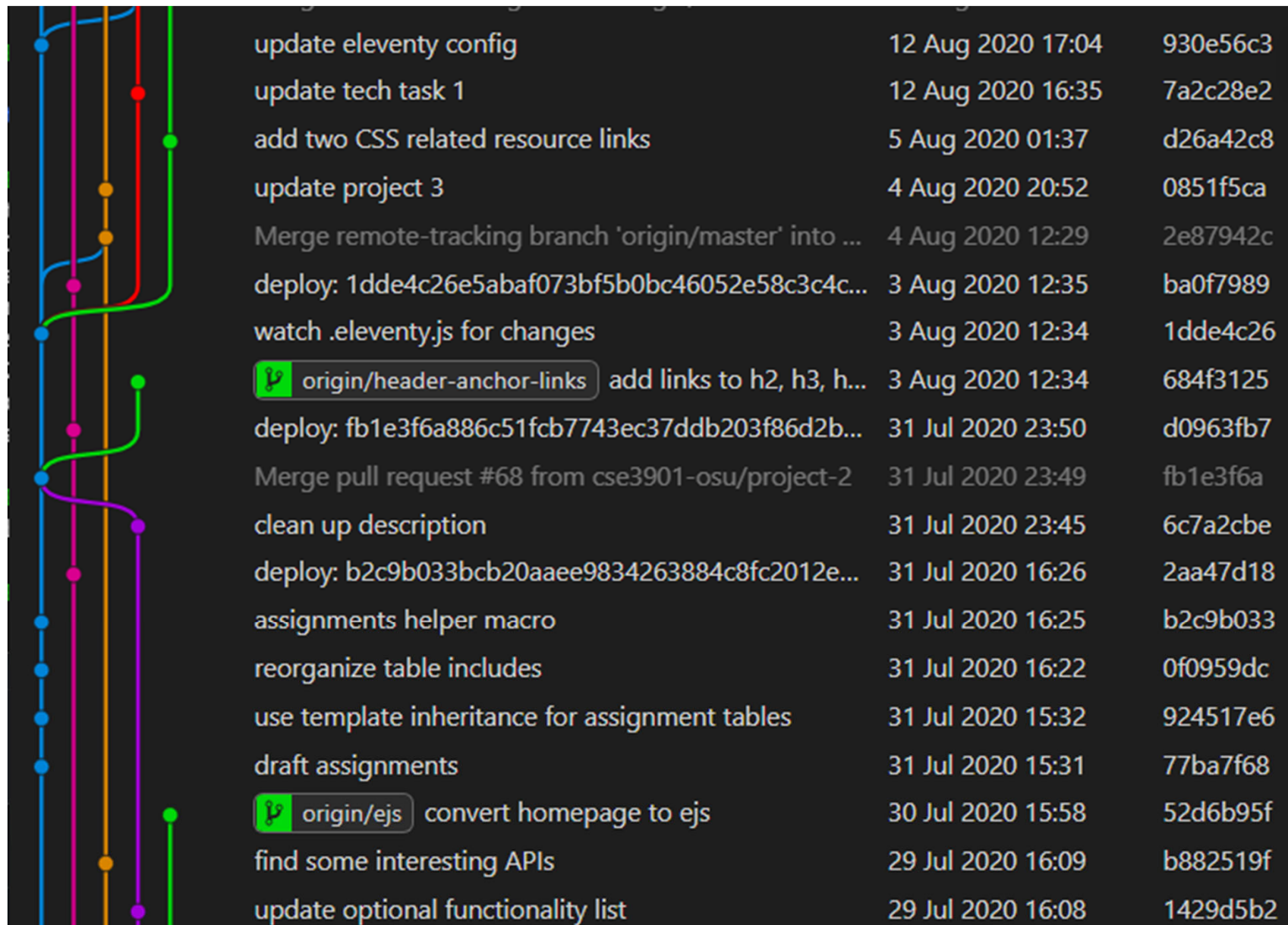
- Every commit (except the first) has 1 or more parents



store



# Example View of DAG



# Example View of DAG

```
$ git log --oneline --no-decorate --graph

* 1618849 clean up css
*   d579fa2 merge in improvements from master
| \
| * 0f10869 replace image-url helper in css
* | b595b10 add buckeye alert notes
* | a6e8eb3 add raw buckeye alert download
| /
* b4e201c wrap osu layout around content
* e9d3686 add Rakefile and refactor schedule loop
* 515aaa3 create README.md
* eb26605 initial commit
```

# Commit

- Each commit is identified by a hash
  - 160 bits (i.e., 40 hex digits)
  - Practically guaranteed to be unique
  - Can use short prefix of hash if unique

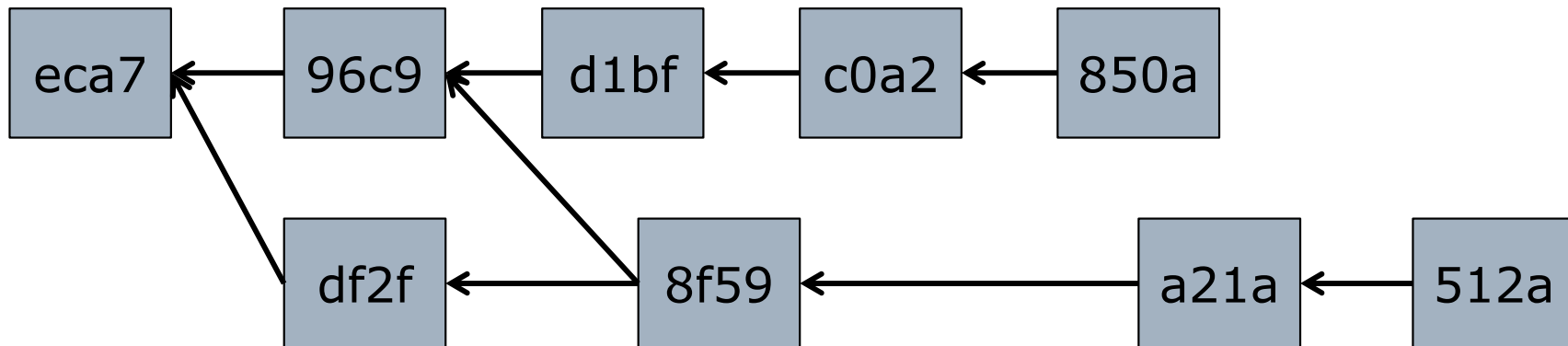
```
$ git show --name-only --no-decorate
commit 16188493c252f6924baa17c9b84a4c1baaed438b
Author: Brutus Buckeye <brutus@users.noreply.github.com>
Date:   Mon Mar 29 15:30:50 2021 +0200
```

```
    clean up css
```

```
source/stylesheets/_site.css
```

# History is a DAG

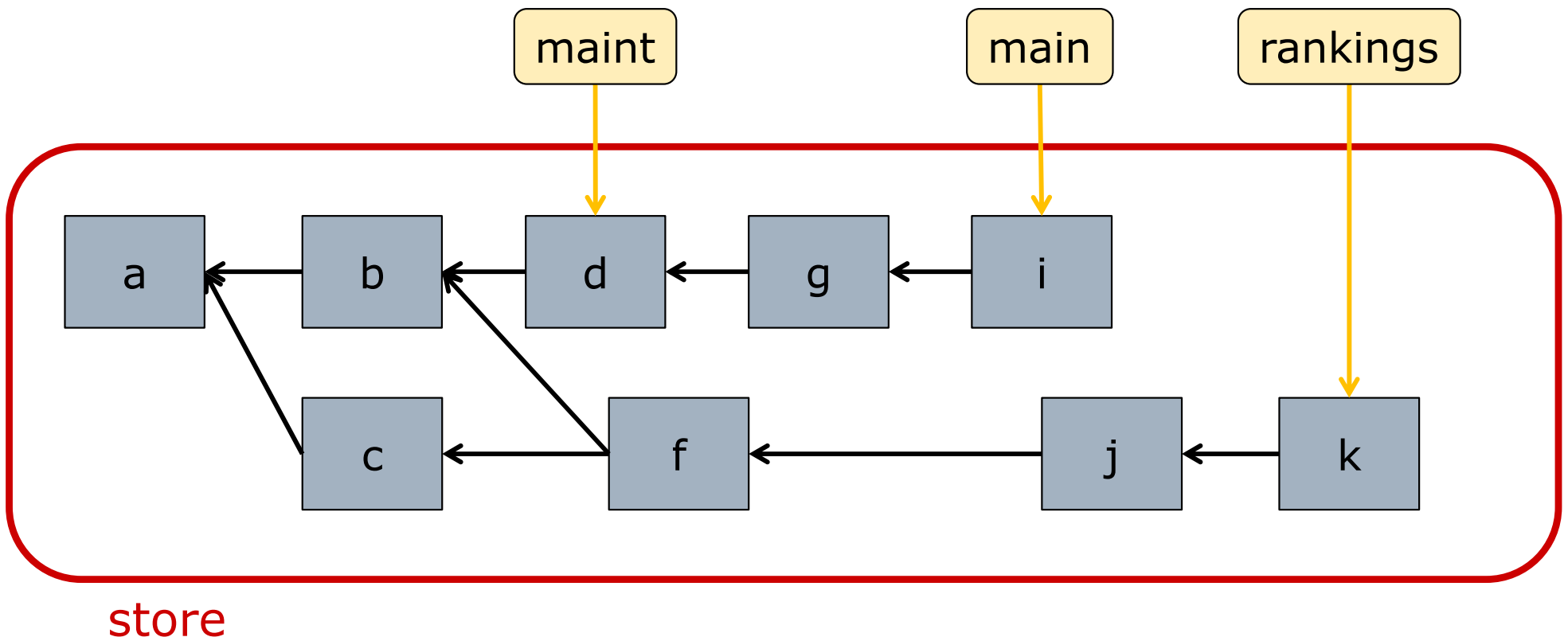
- A better picture would label each commit with its hash (prefix)



- But in these slides, we abbreviate the hash id's as just: 'a', 'b', 'c'...

# Nomenclature: Branch

- ❑ *Branch*: a pointer to a commit
- ❑ Different from “branch” in DAG's shape

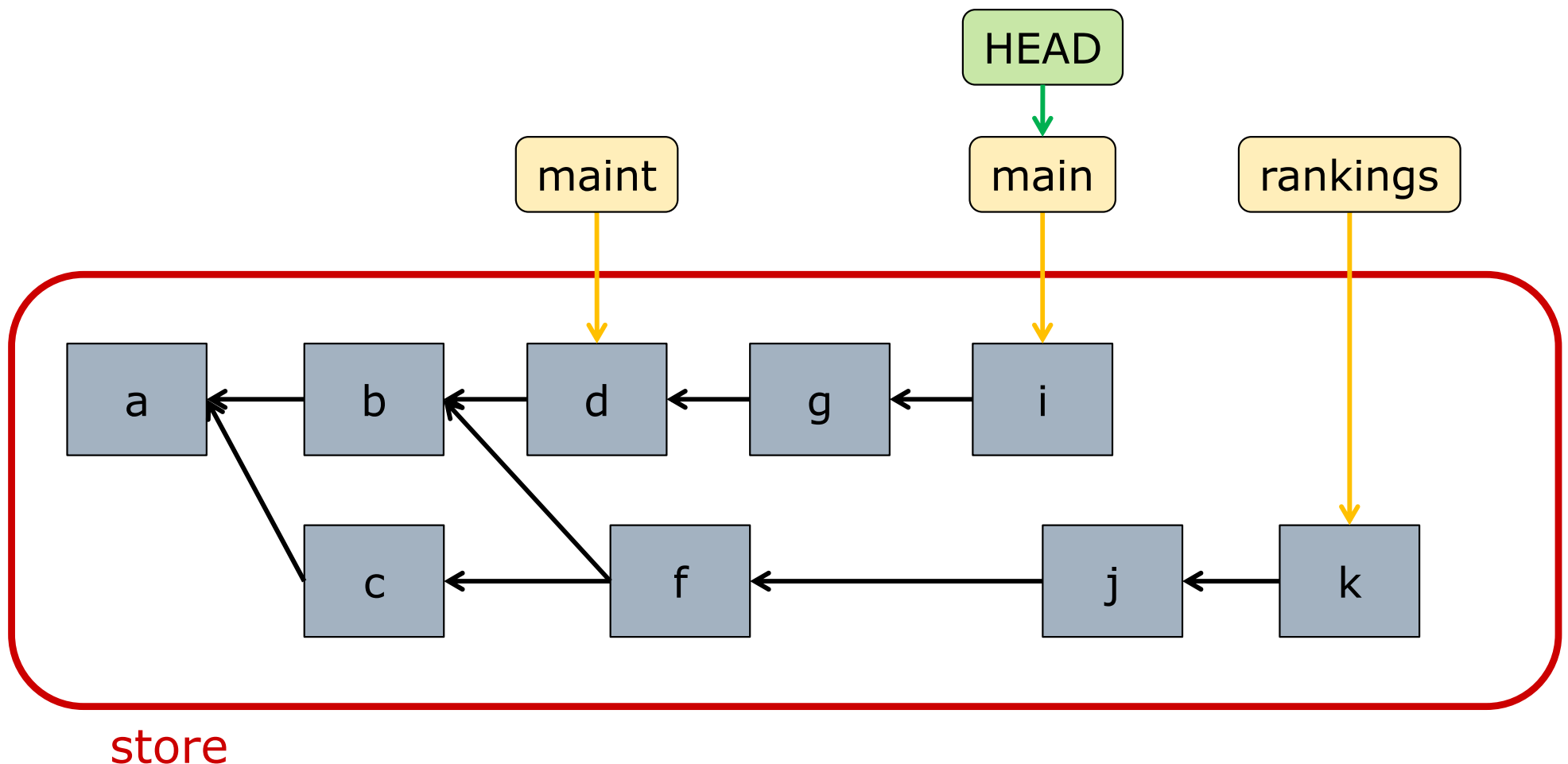


# A Note on Changing Defaults

- Any name can be used for a branch
  - Typically short, but hopefully descriptive
  - Many branches, each with a unique name
- Initially, a repo has a single branch
  - Repos created on GitHub use “main” as the initial branch name
  - Repos created locally (git 2.42) still use old initial branch name (“master”)
    - This default is user-configurable! (as of git 2.28, 7/27/20)

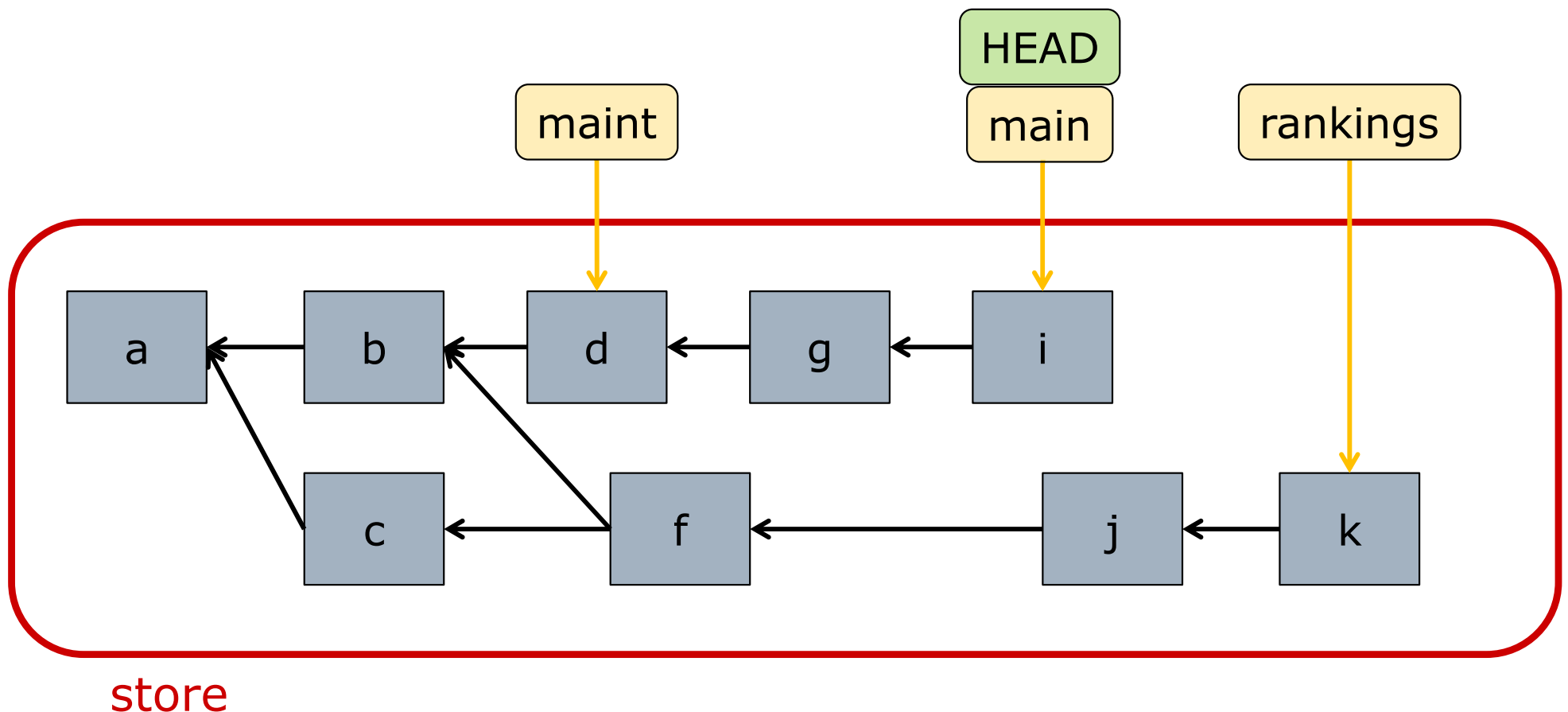
# Nomenclature: HEAD

- *HEAD*: a special reference, (usually) points to a branch



# Nomenclature: HEAD

- Useful to think of HEAD as being “attached” to a particular branch



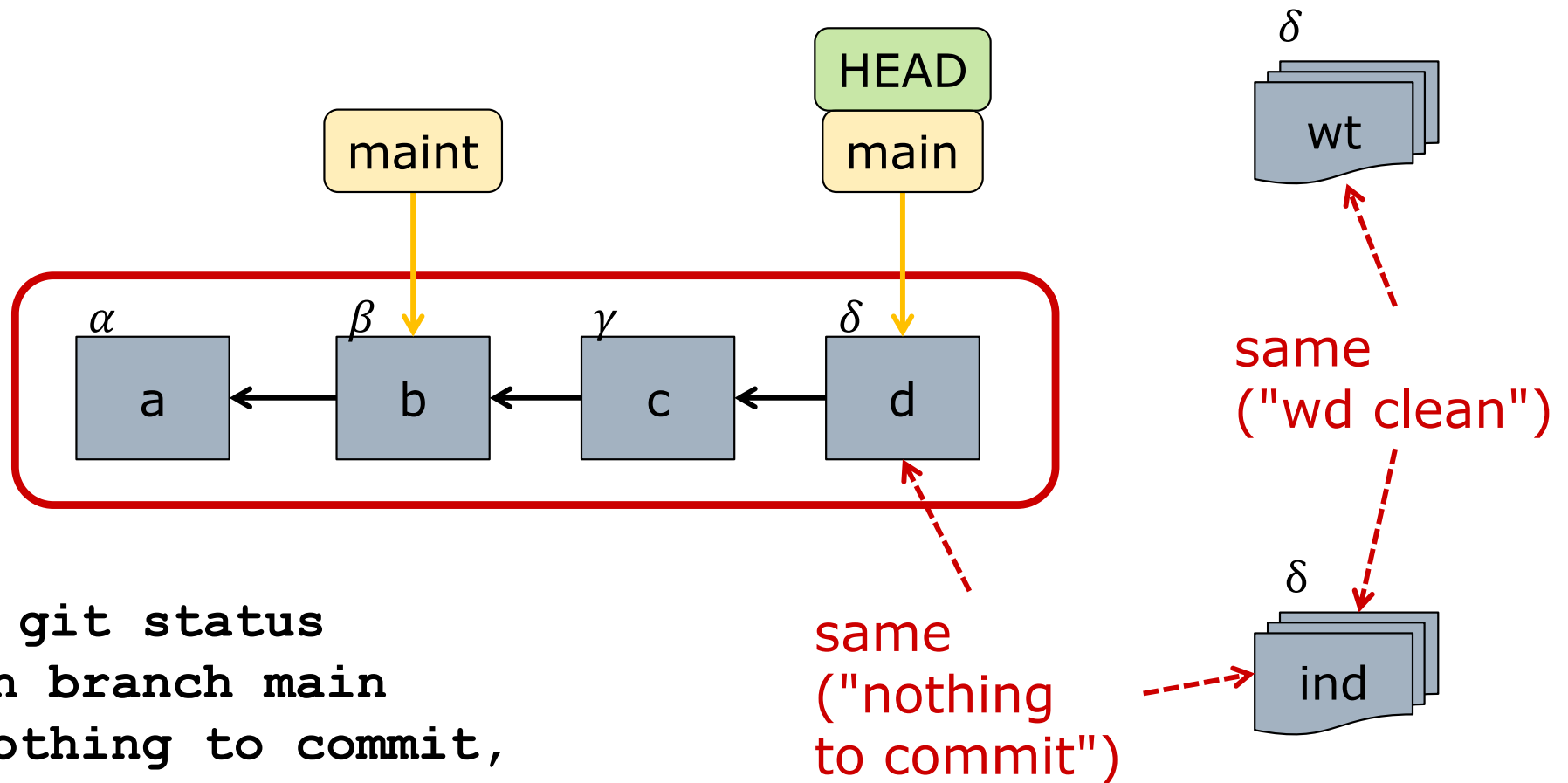


# View of DAG with Branches

```
$ git log --oneline --graph
```

```
* 1618849 (HEAD -> main) clean up css
* d579fa2 (alert) merge in improvements from master
|\
| * 0f10869 replace image-url helper in css
* | b595b10 add buckeye alert notes
* | a6e8eb3 add raw buckeye alert download
|/
* b4e201c wrap osu layout around content
* e9d3686 add Rakefile and refactor schedule loop
* 515aaa3 create README.md
* eb26605 initial commit
```

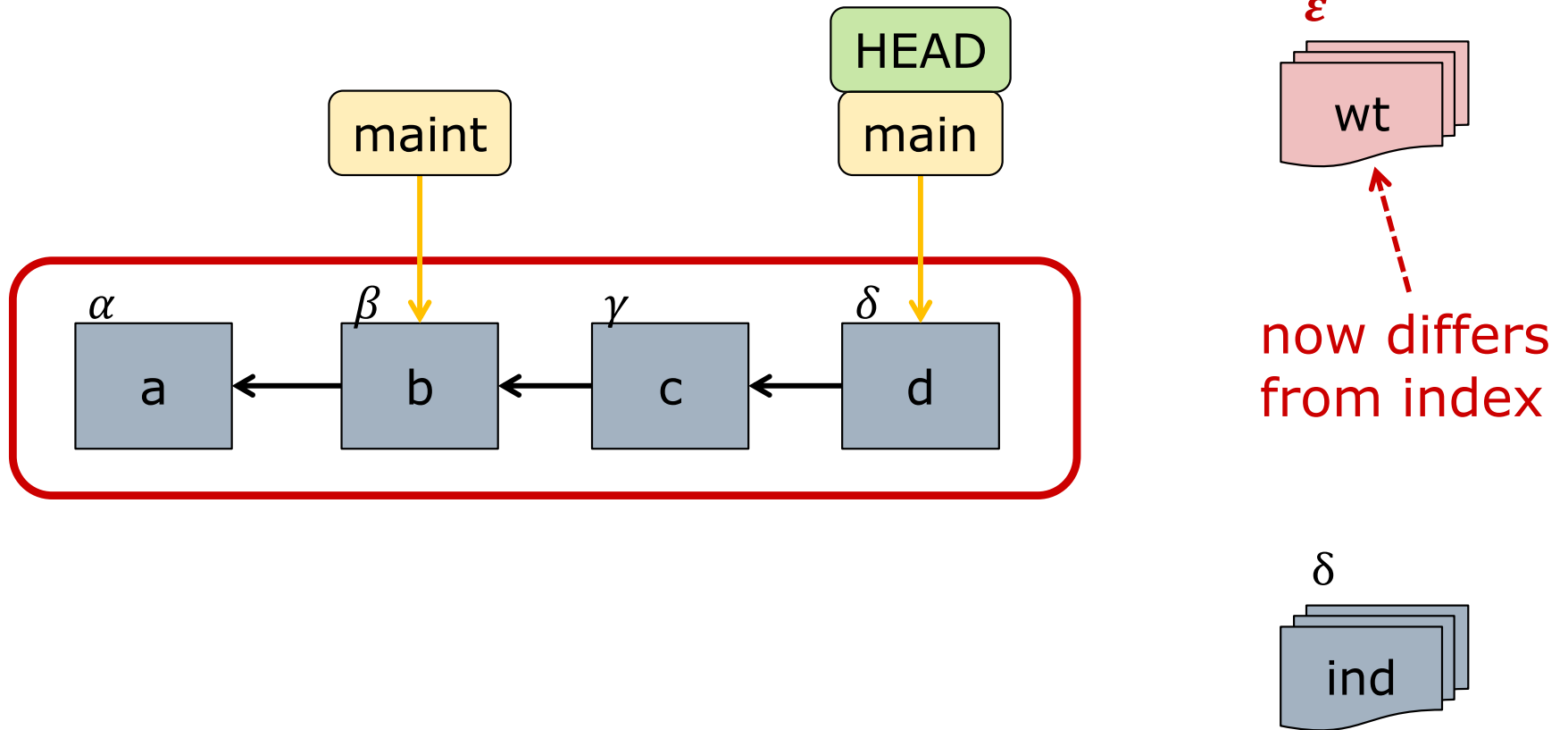
# A "Clean" Repository



```
$ git status
On branch main
nothing to commit,
working directory clean
```

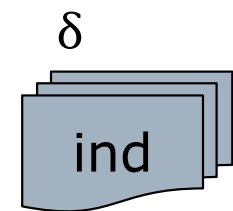
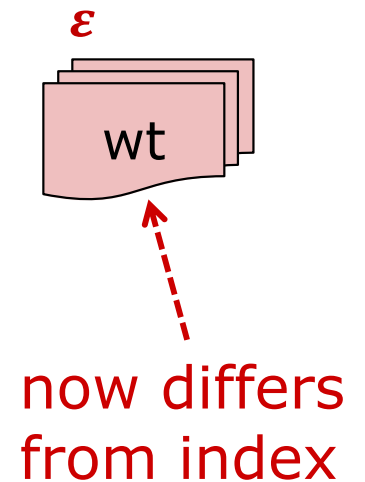
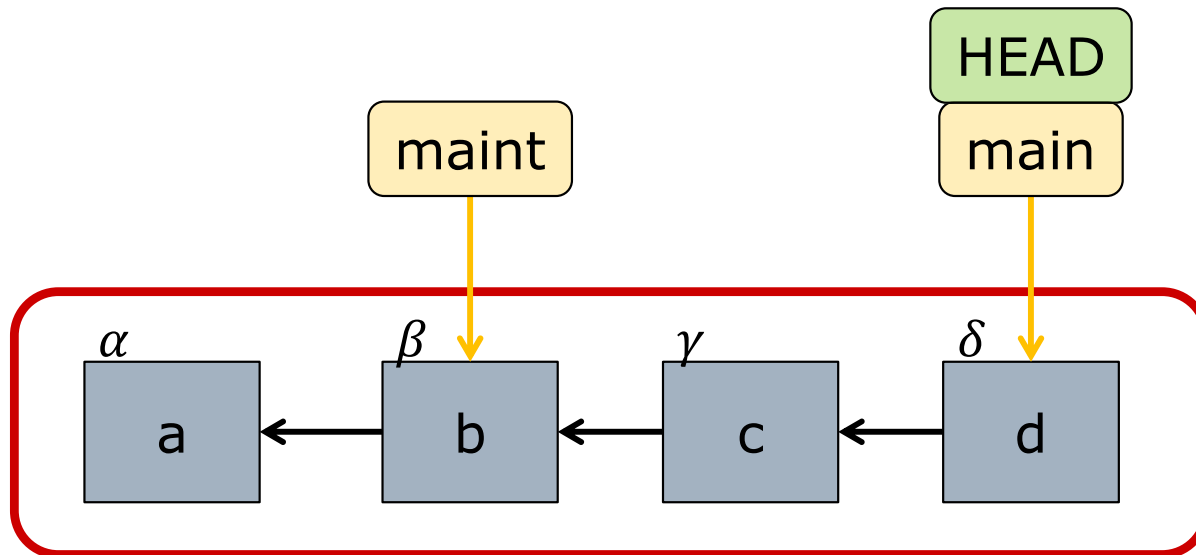
# Edit Files in Working Tree

- Add files, remove files, edit files...



# Edit Files in Working Tree

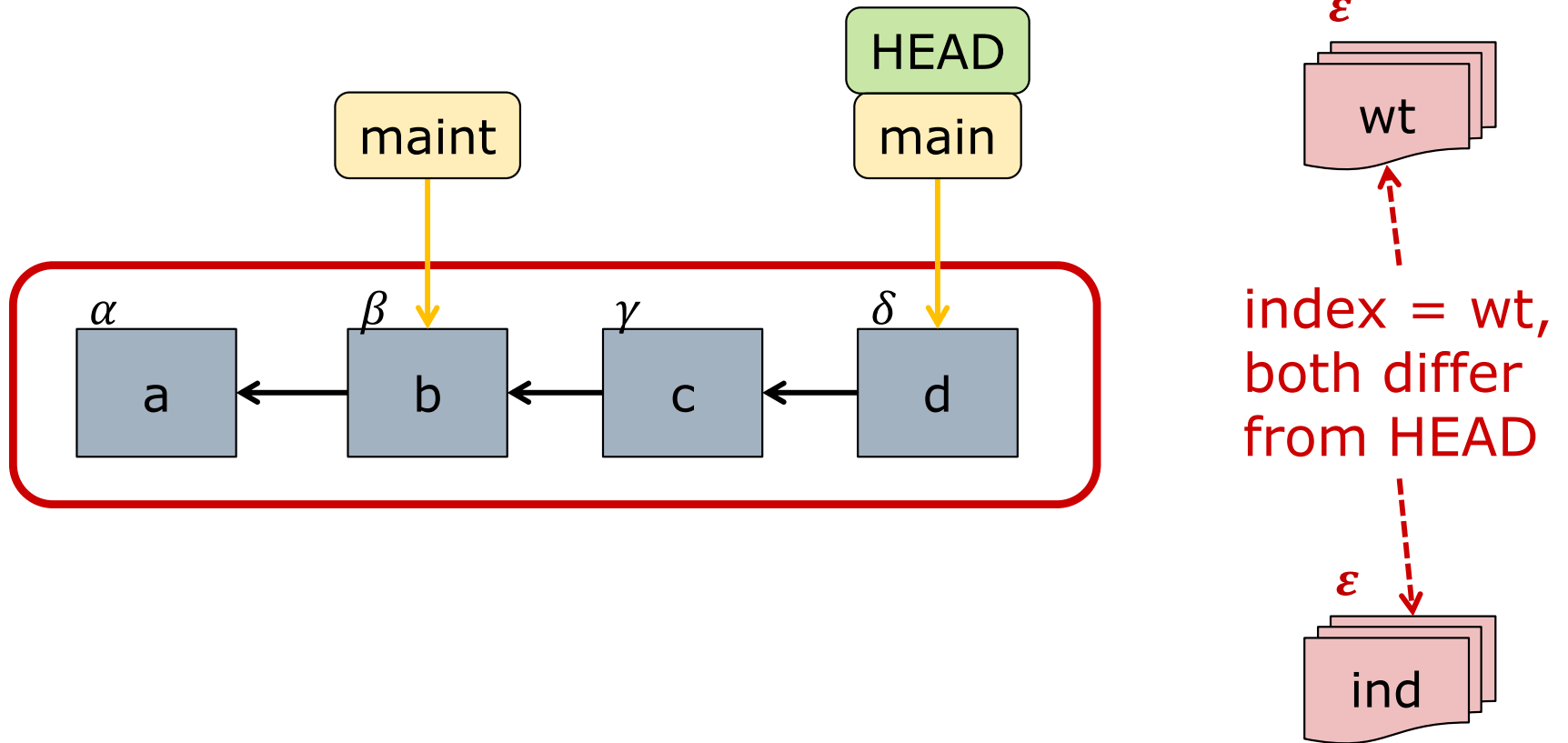
- Add files, remove files, edit files...



```
$ git status
On branch main
Changes not staged for commit:
  modified:  css/demo.css
```

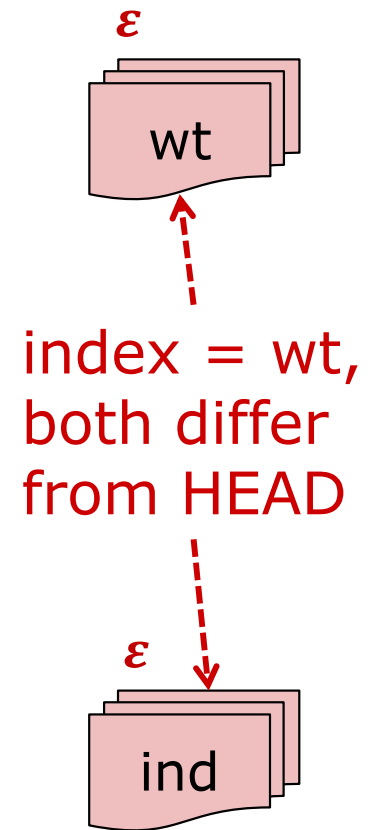
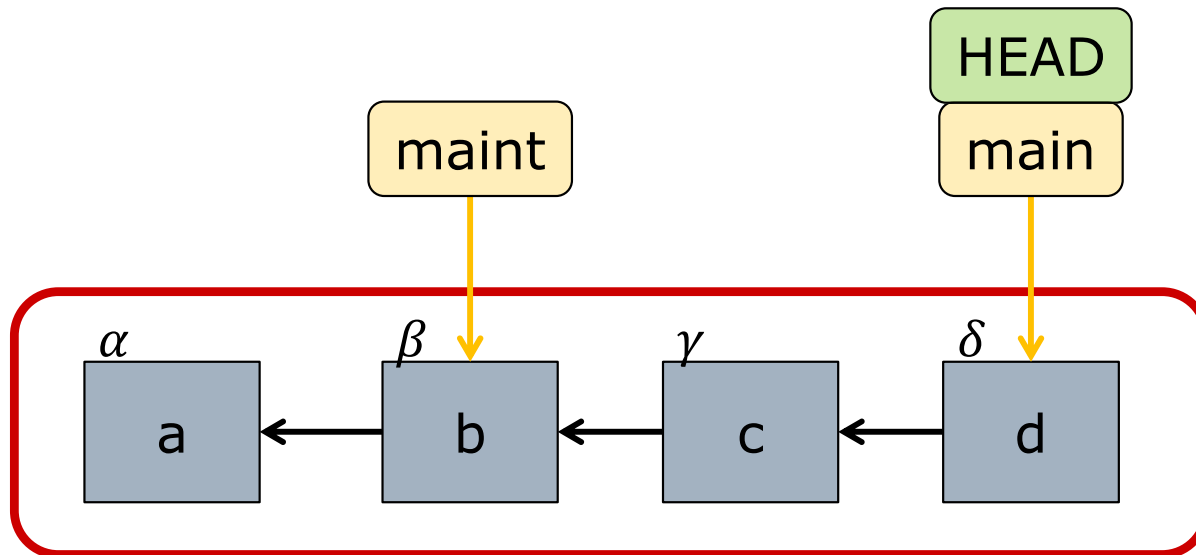
# Add: Working Tree $\rightarrow$ Index

`$ git add . # current directory, and below`



# Add: Working Tree $\rightarrow$ Index

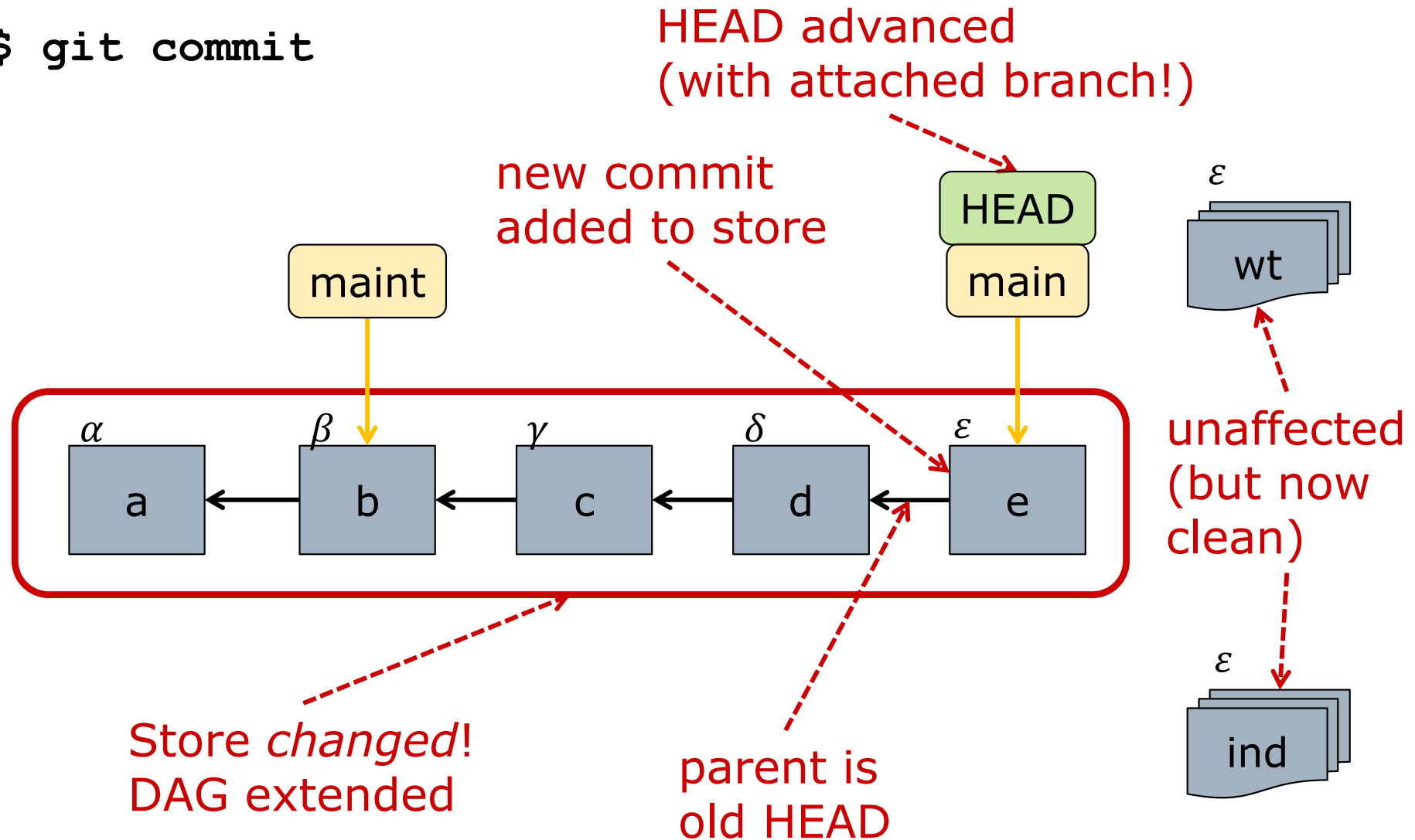
```
$ git add . # current directory, and below
```



```
$ git status
On branch main
Changes to be committed:
  modified:  css/demo.css
```

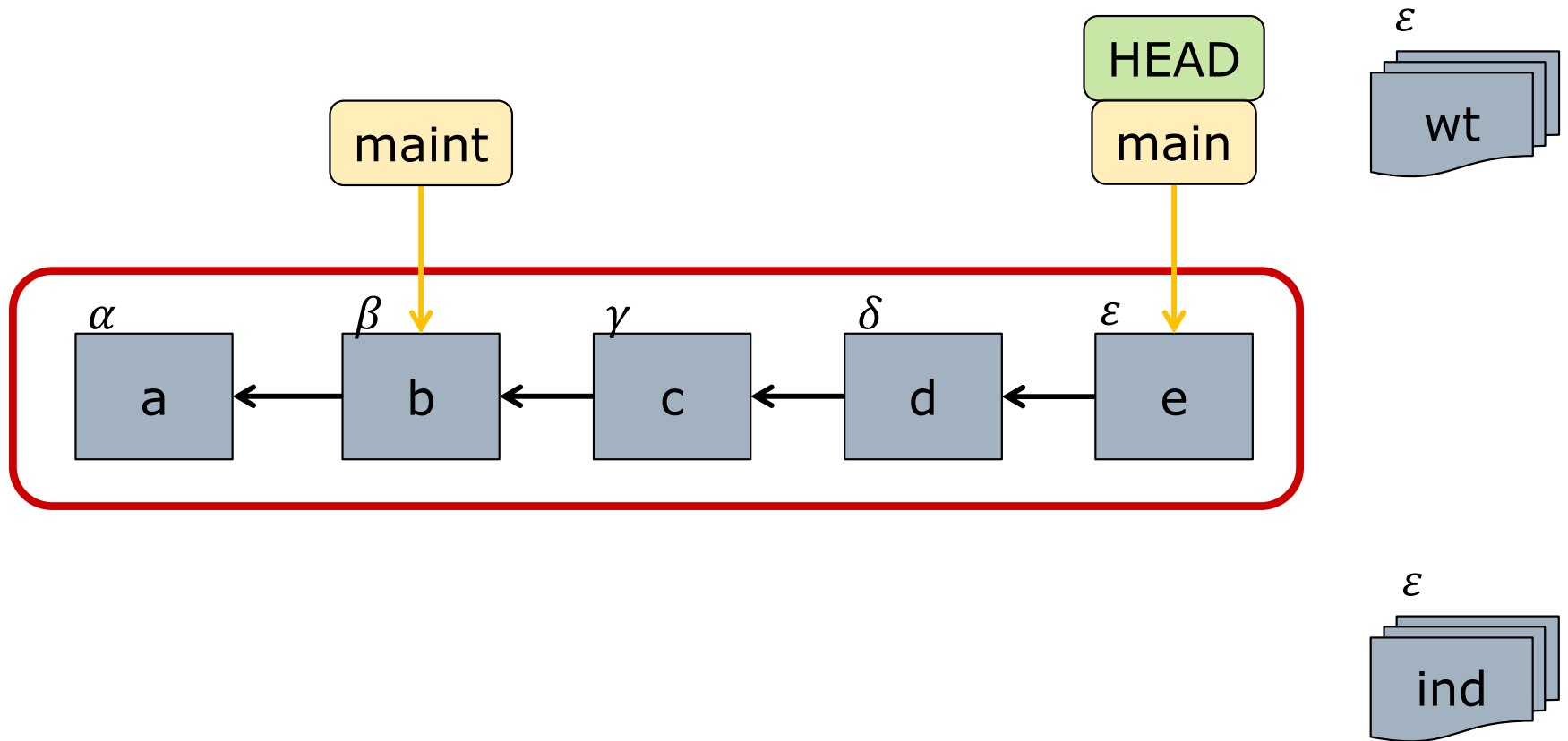
# Commit: Index → Store

```
$ git commit
```



# The (New) State of Repository

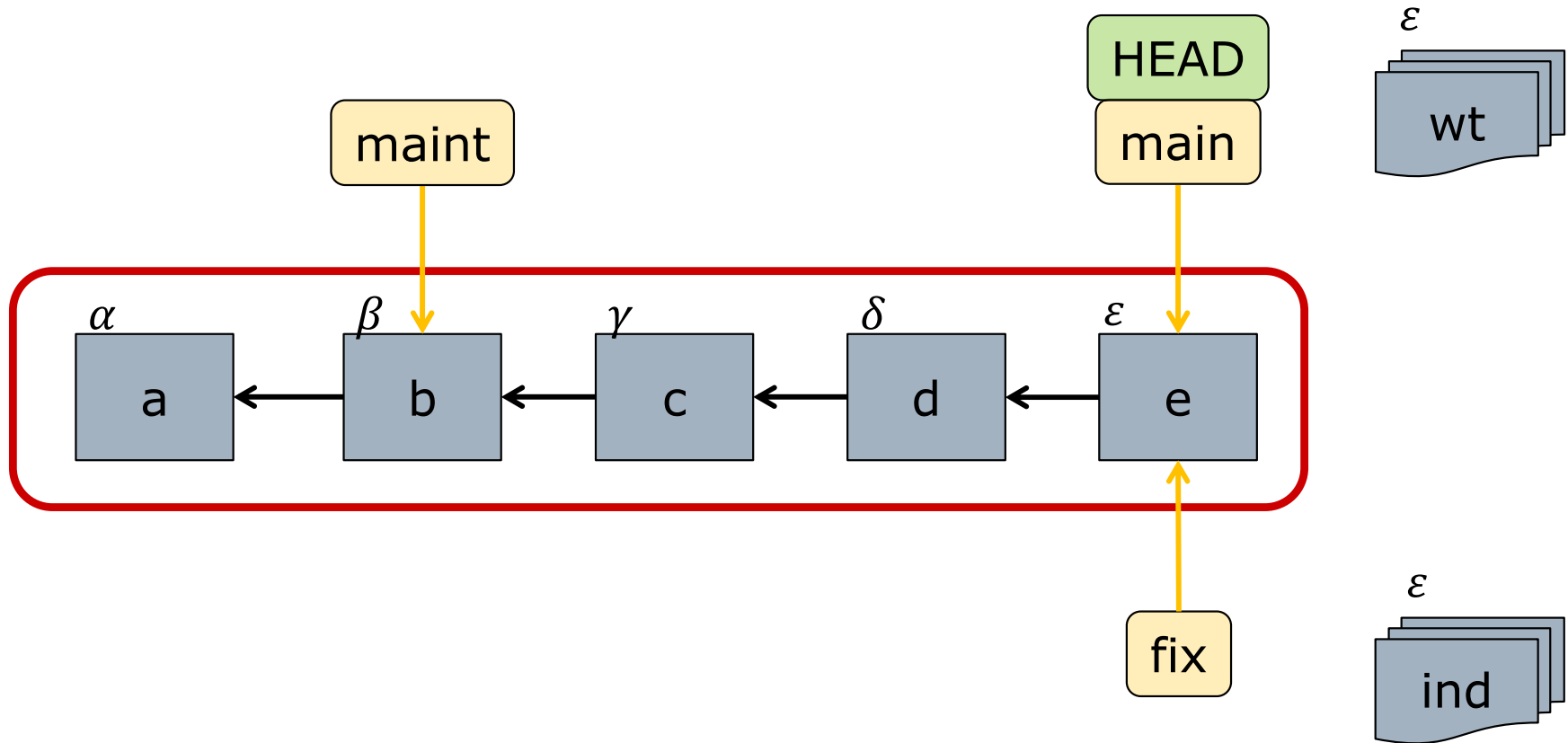
Computer Science and Engineering ■ The Ohio State University





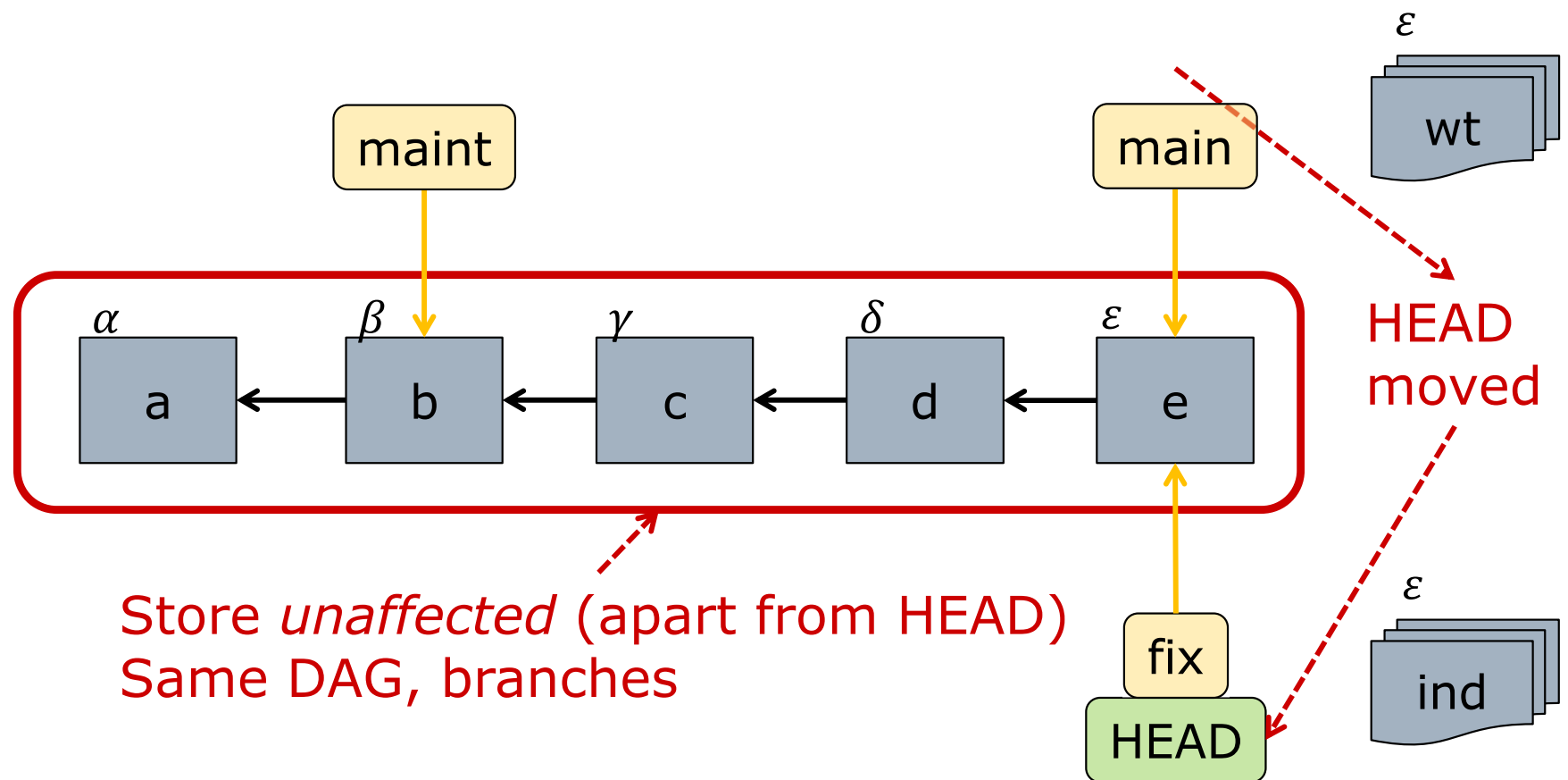
# Creating a New Branch

```
$ git branch fix
```



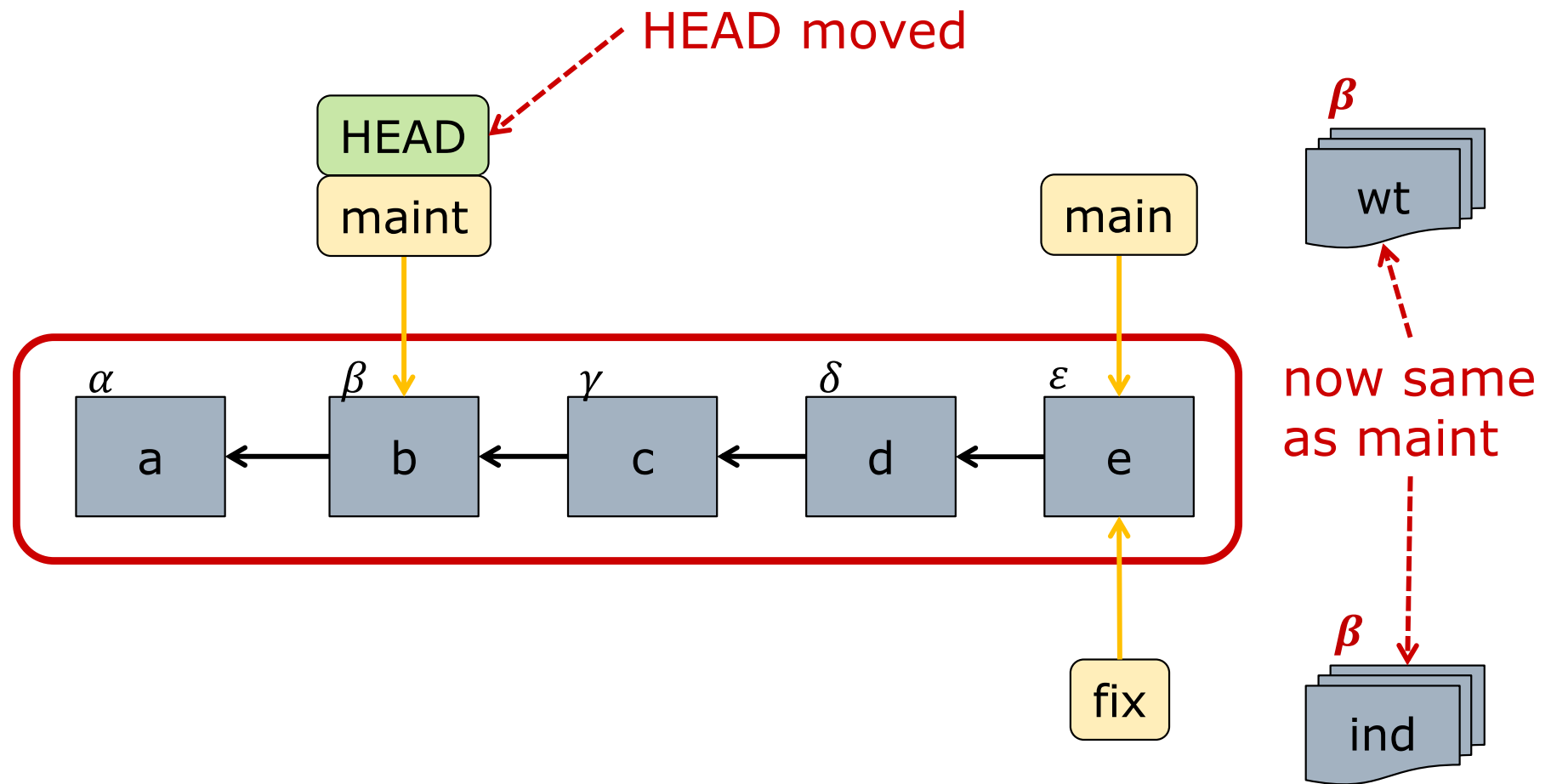
# Checkout: Changing Branch

```
$ git checkout fix
```



# Checkout: Changing Branch

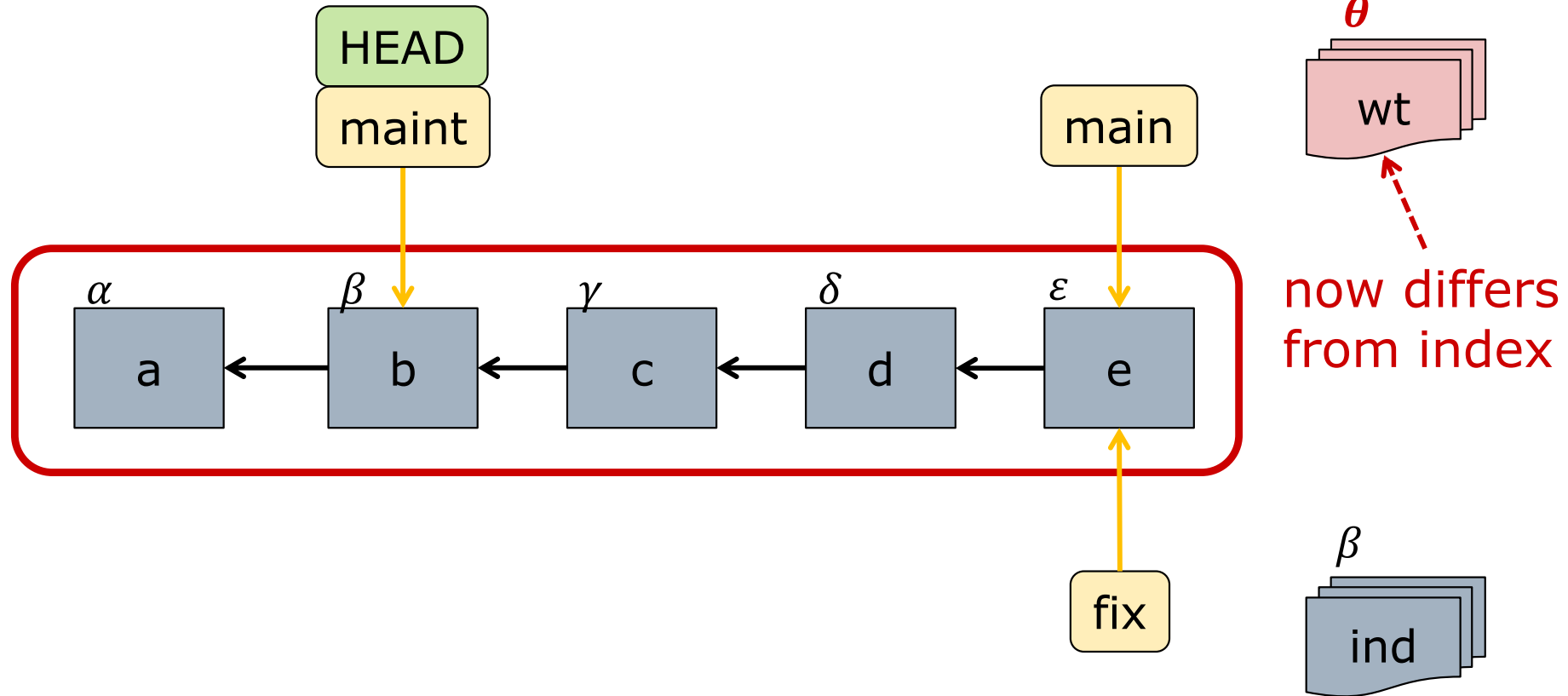
```
$ git checkout maint
```



□ Advice: checkout <branch> *only* when wt is clean

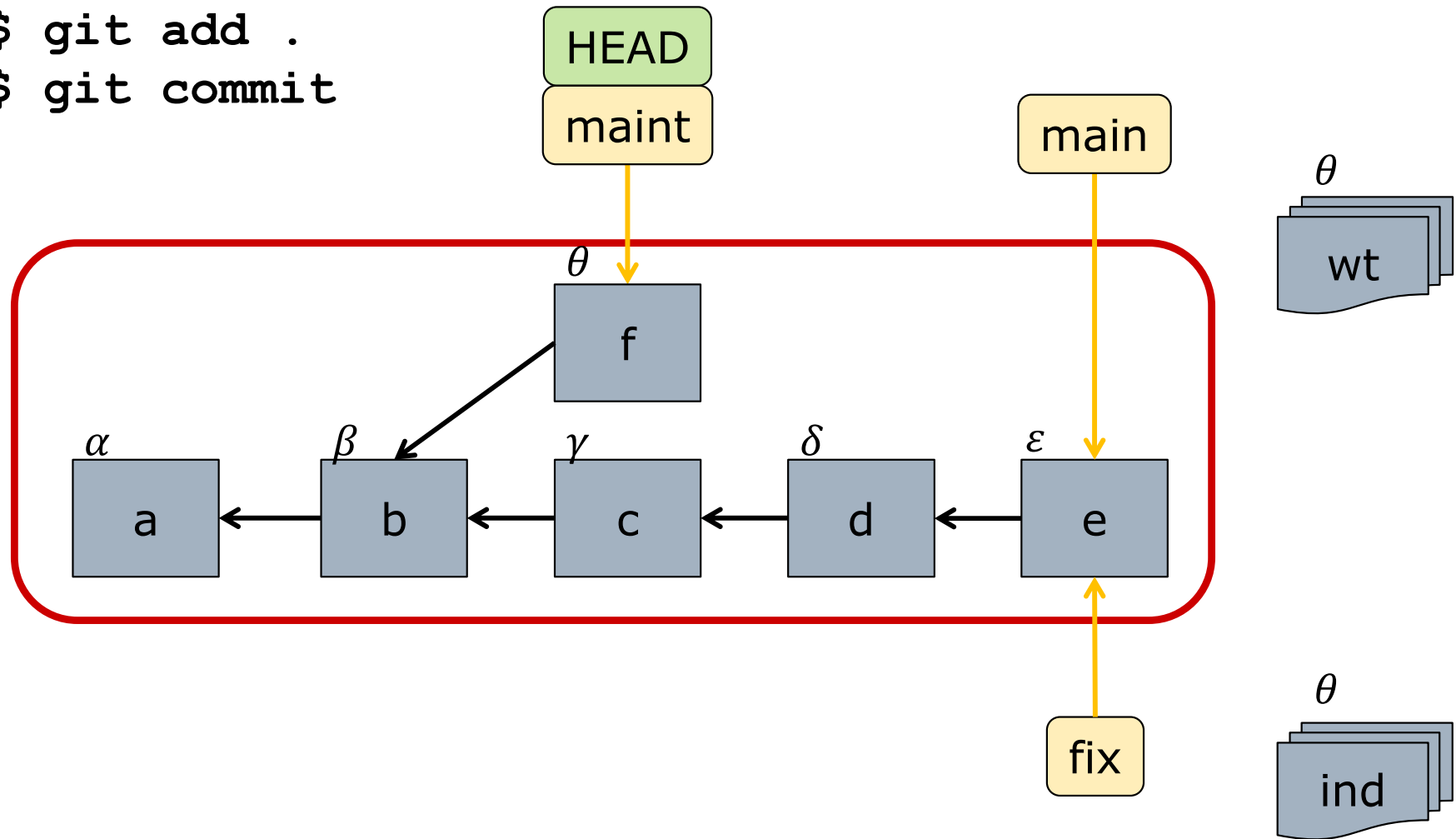
# Edit Files in Working Tree

- Add files, remove files, edit files...



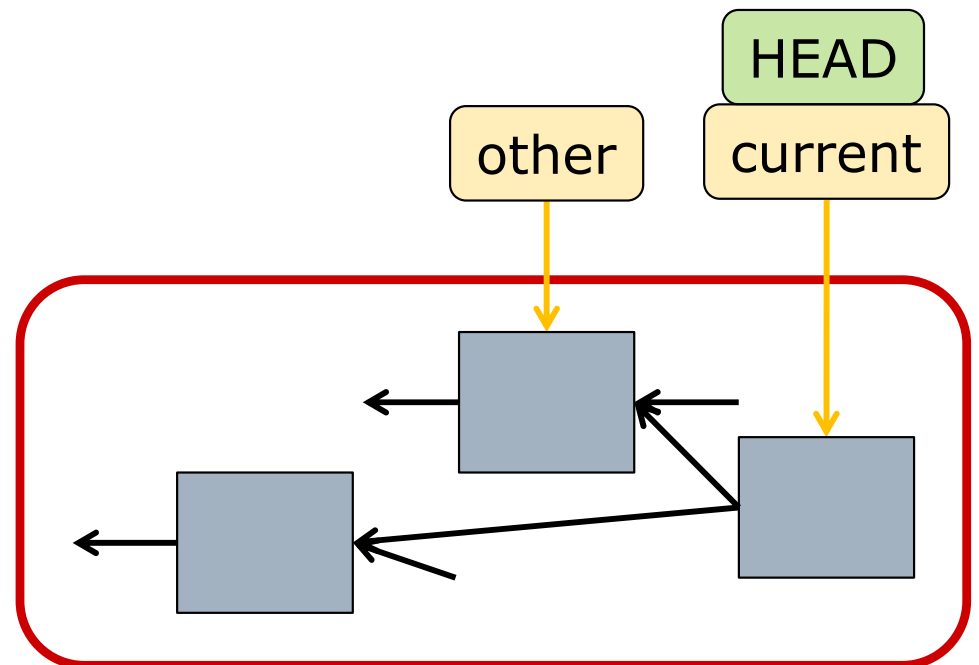
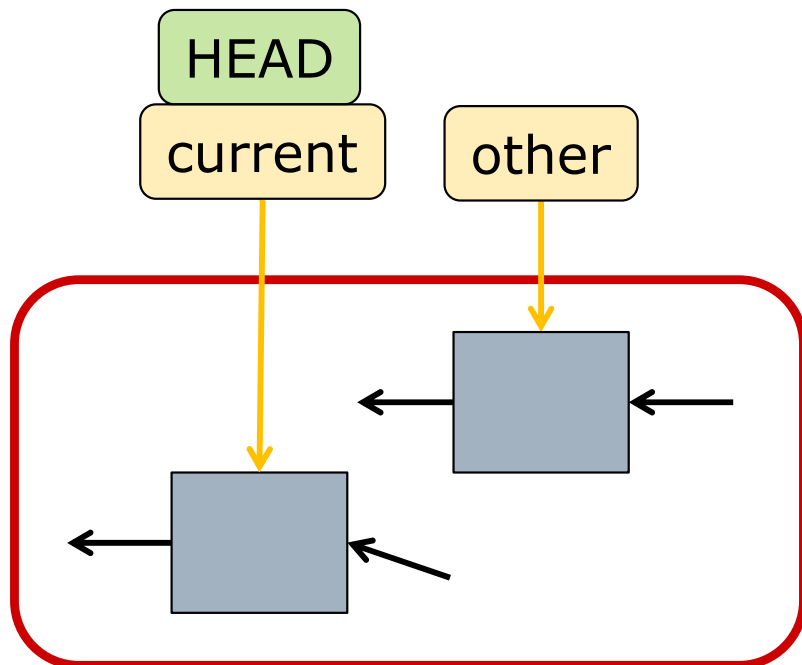
# Add & Commit: Update Store

```
$ git add .  
$ git commit
```



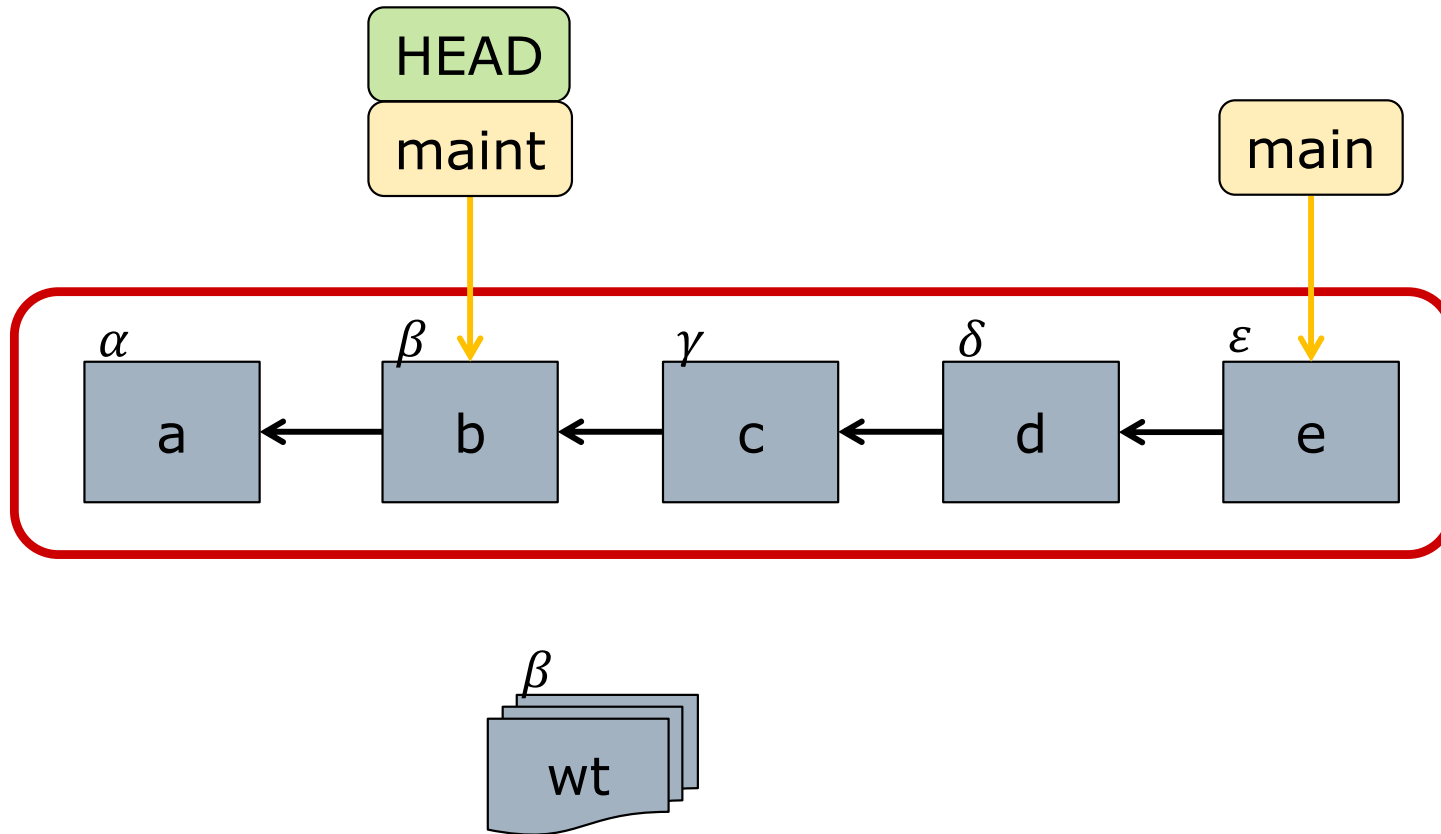
# Merge: Bringing History together

- Bring work from another branch into current branch
  - Implemented features, fixed bugs, etc.
- Updates current branch, not other



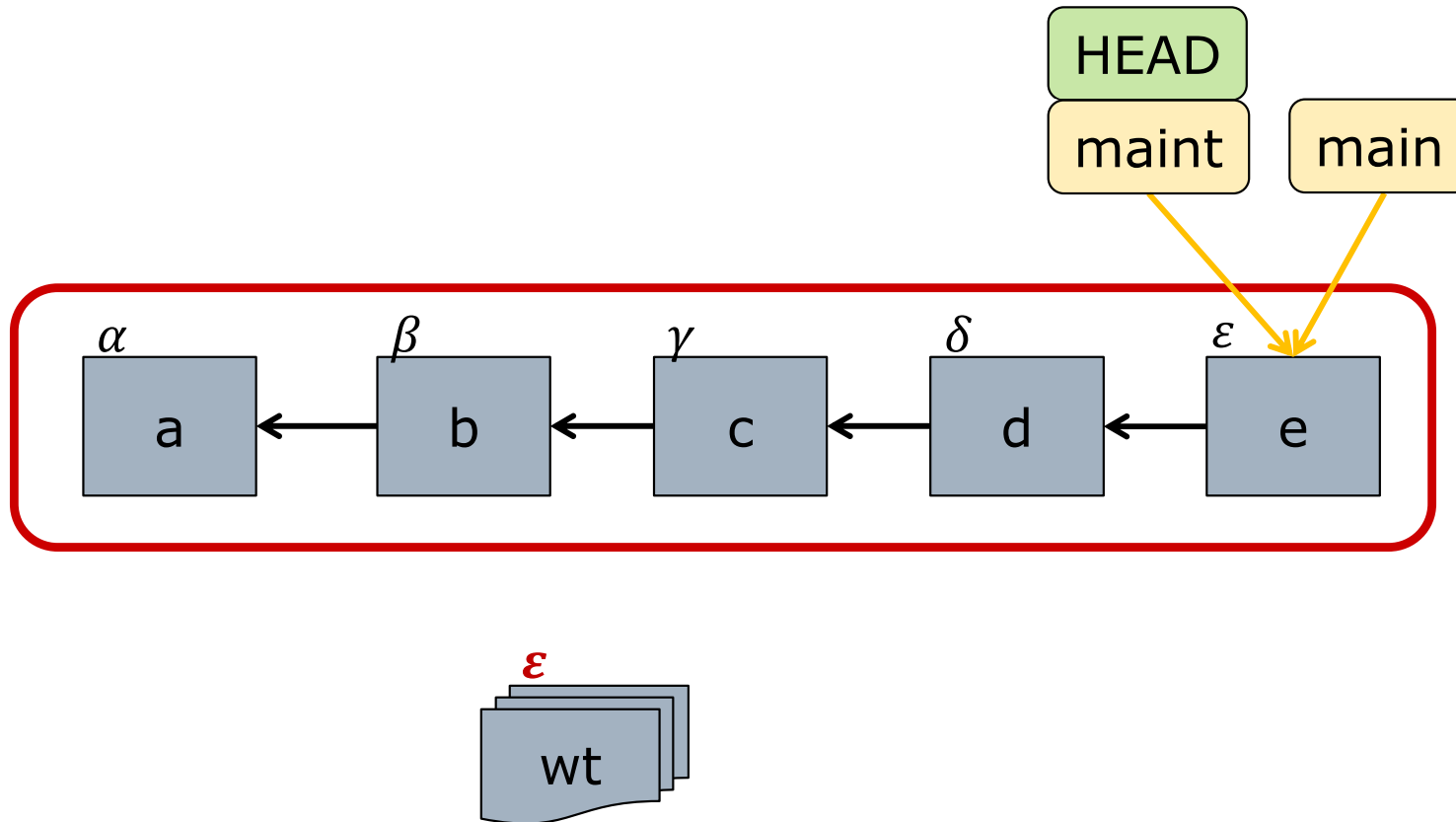
# Merge – Case 1: Ancestor

- HEAD is an ancestor of other branch



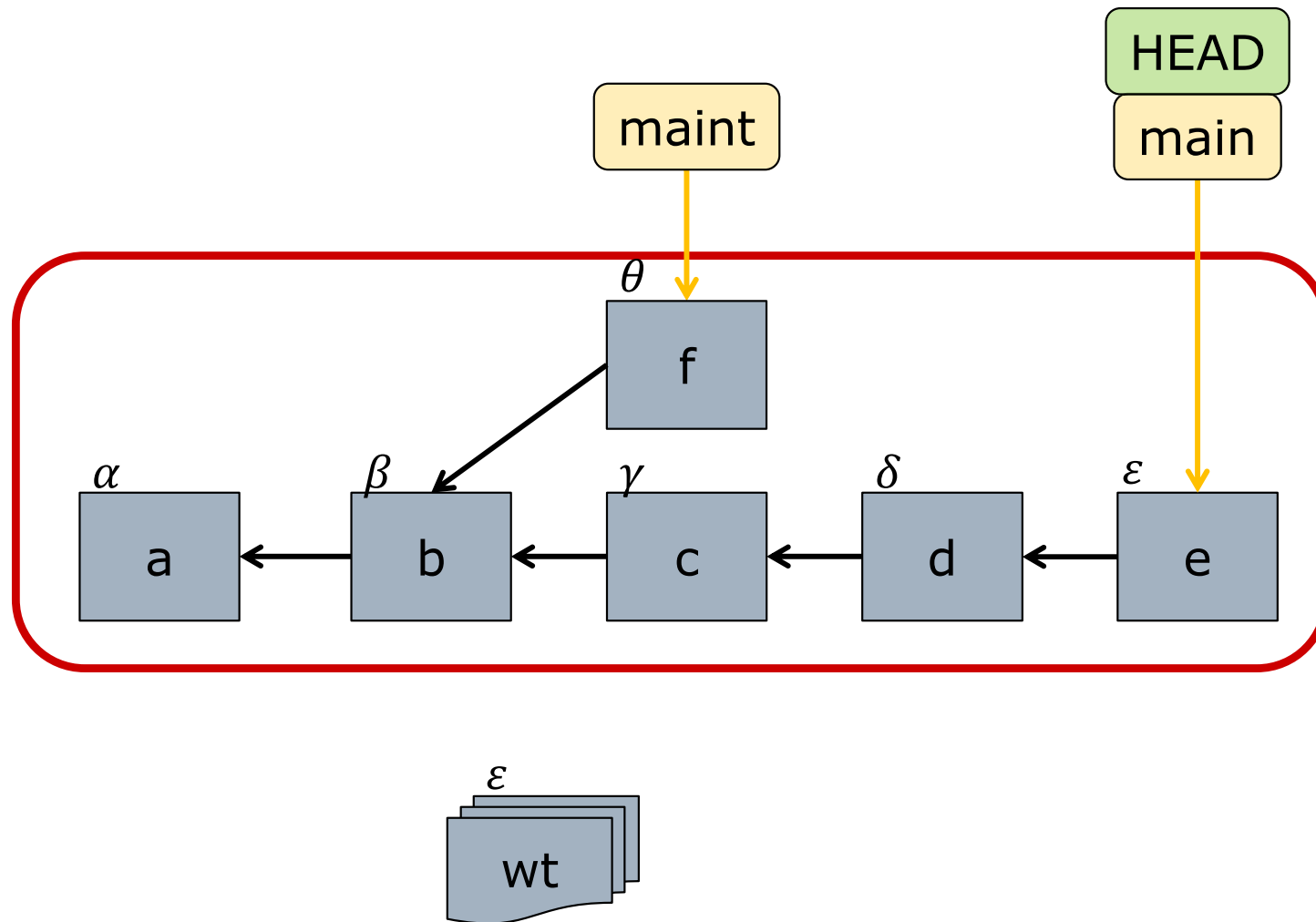
# Fast-Forward Merge

```
$ git merge main
```



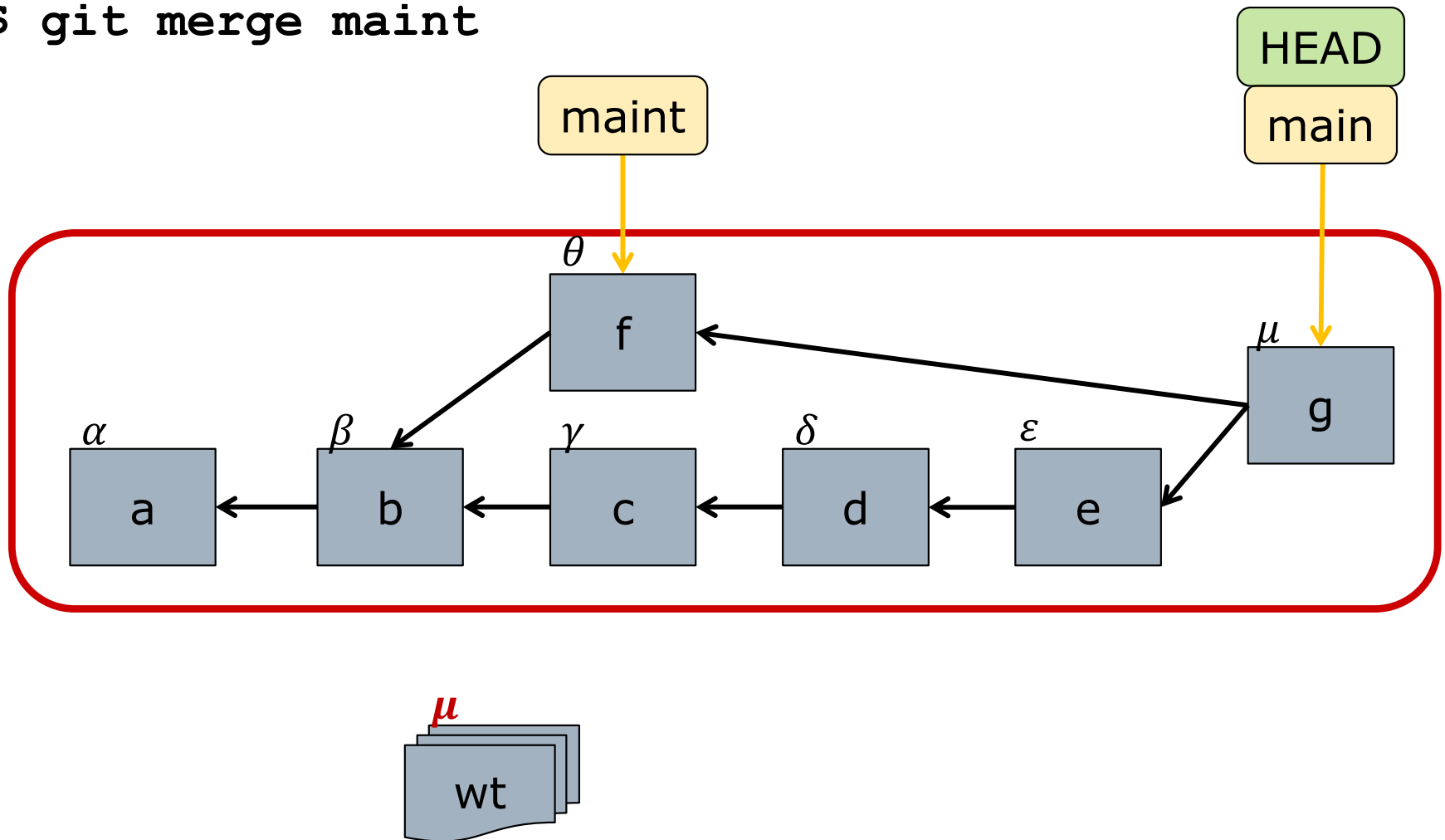


# Merge – Case 2: No Conflicts



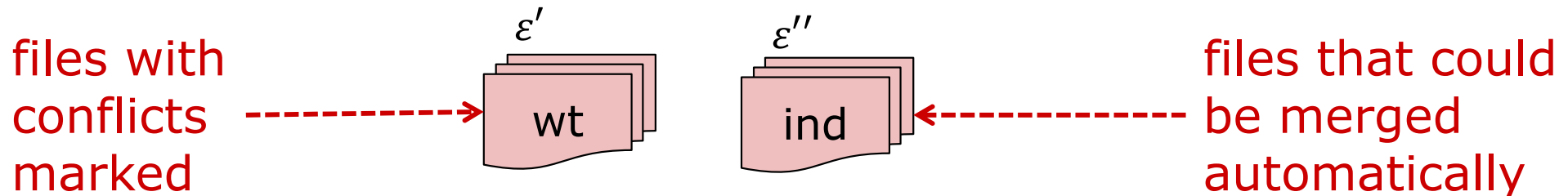
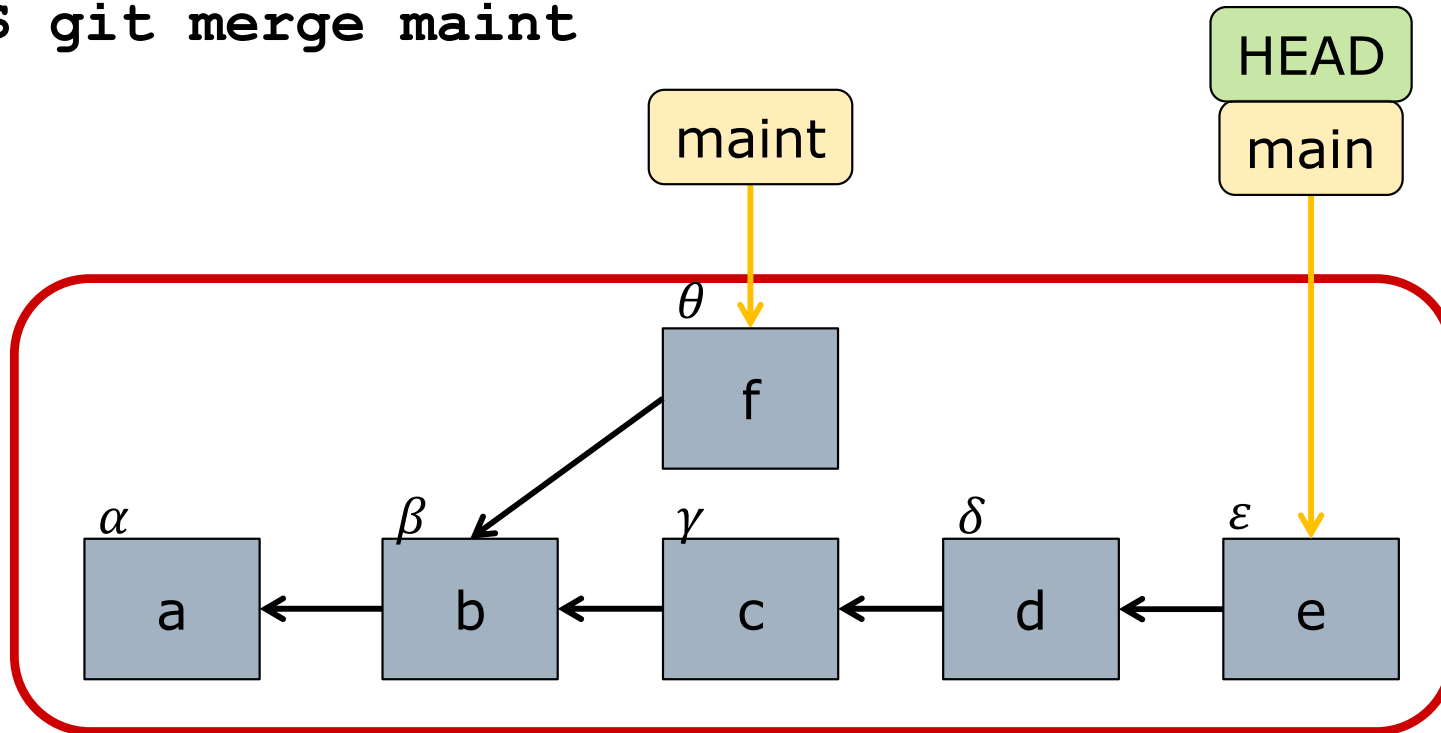
# Merge Automatically Commits

```
$ git merge maint
```



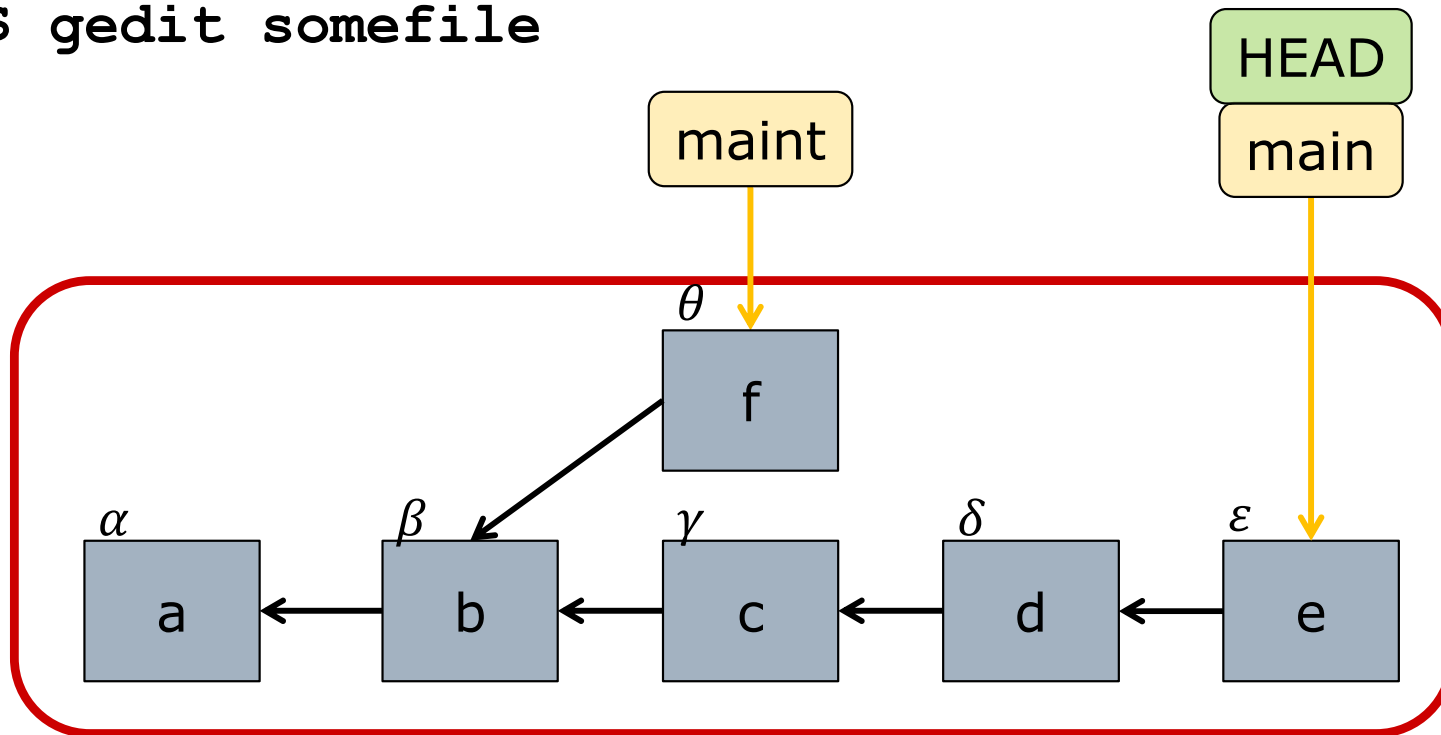
# Merge – Case 3: Conflicts Exist

```
$ git merge maint
```

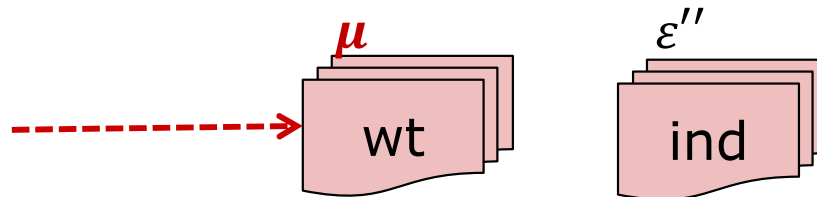


# Merge: Resolve Conflicts

`$ gedit somefile`

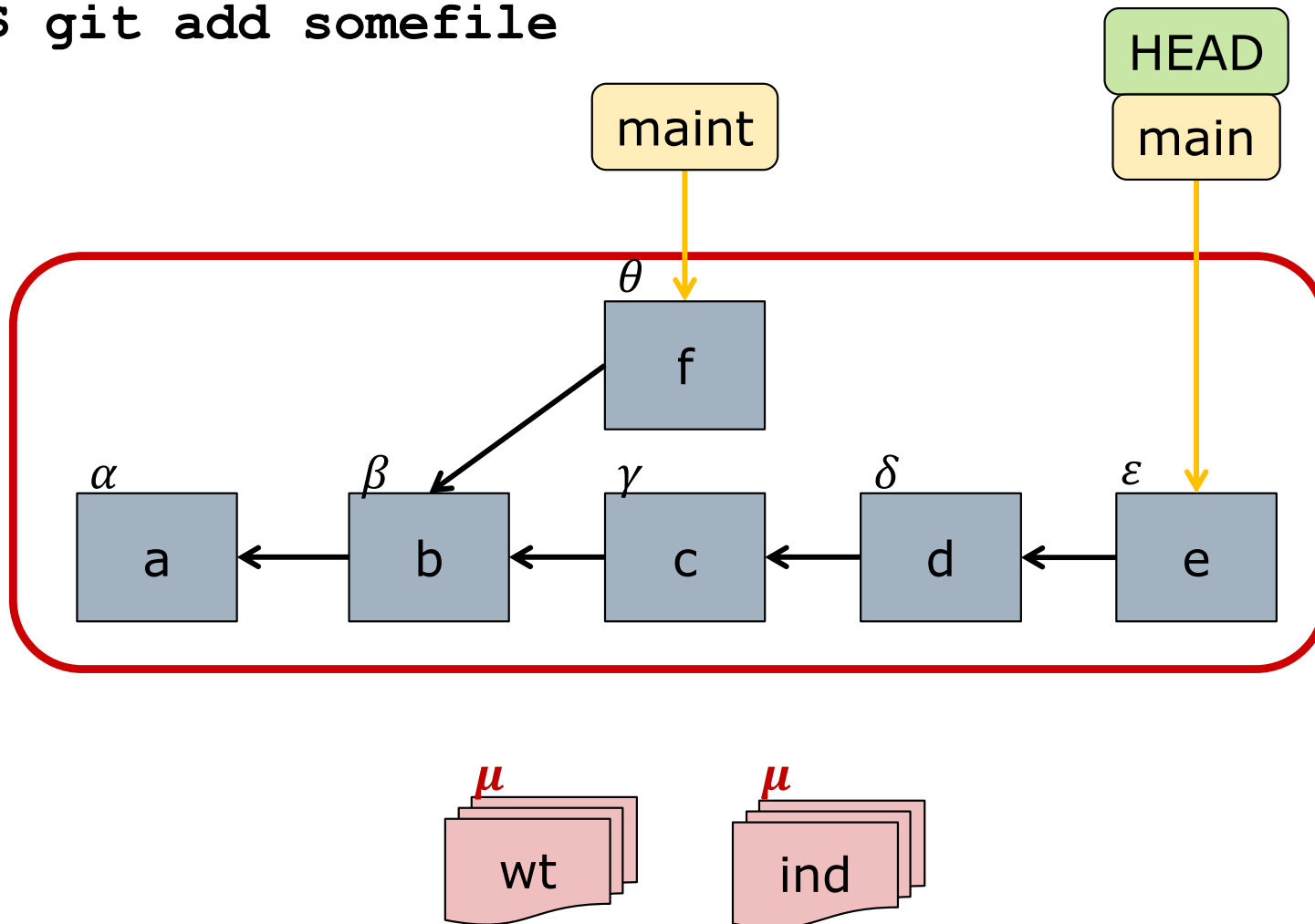


files with  
conflicts  
resolved



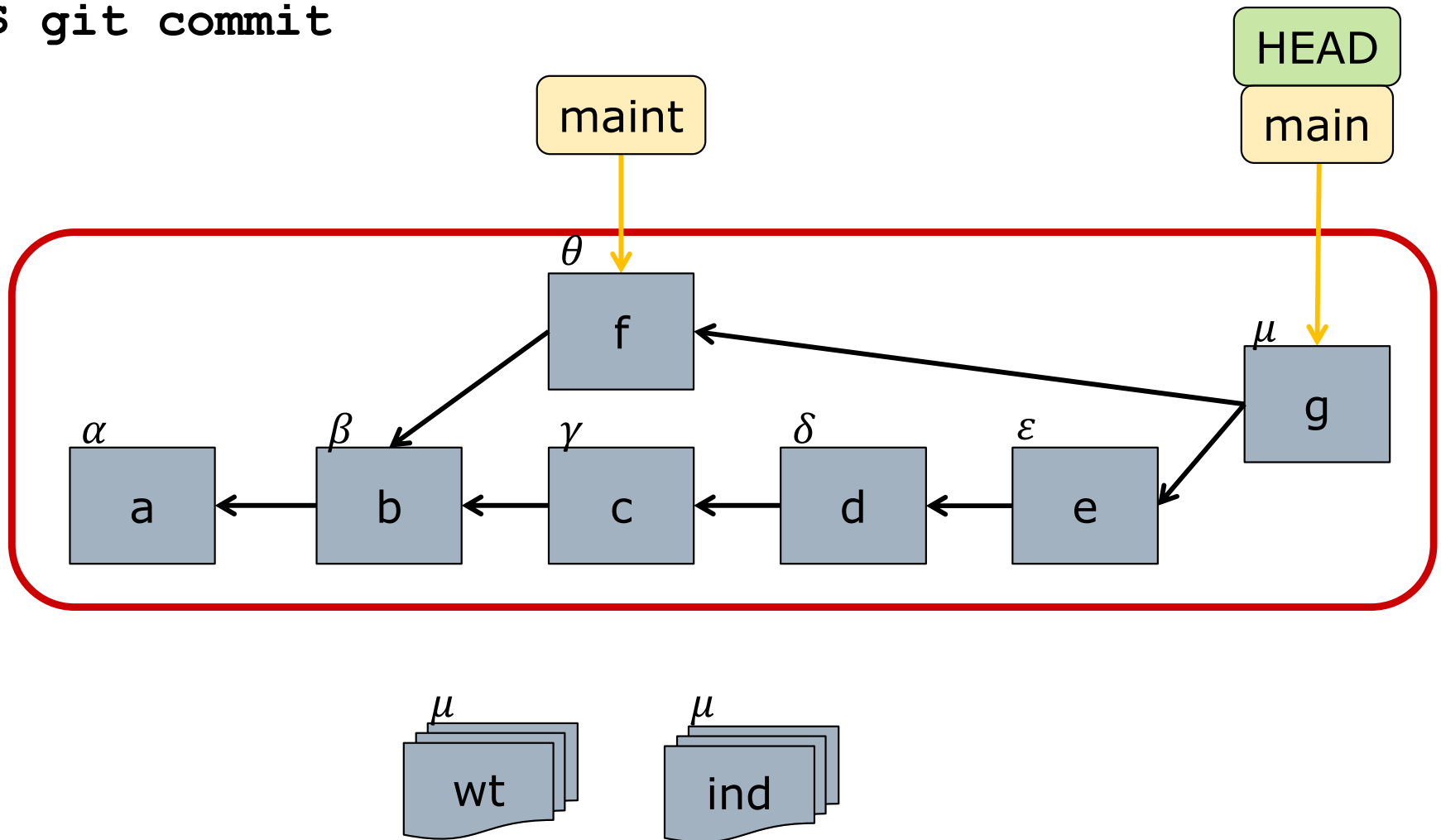
# Merge with Conflicts: Add

```
$ git add somefile
```



# Merge with Conflicts: Commit

```
$ git commit
```



# Merge: Edit to Resolve Conflicts

```
13
14  /**
15  | * Prints the welcome message
16  */
17  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
18  <<<<<< HEAD (Current Change)
19  function printMessage(showUsage, message) {
20      console.log(message);
21  }
22  =====
23  function printMessage(showUsage, showVersion) {
24      console.log("Welcome To Line Counter");
25      if (showVersion) {
26          console.log("Version: 1.0.0");
27      }
28  }
29  >>>>>> theirs (Incoming Change)
30  if (showUsage) {
31      console.log("Usage: node base.js <file1> <file2> ... ");
32  }
33  }
```

Resolve in Merge Editor

⚙ You, 20 seconds ago Ln 11, Col 26 Spaces: 4 UTF-8 CRLF {} JavaScript

# Merge: 3-way Merge Editor

The screenshot displays the Visual Studio Code interface during a 3-way merge of `target.js`. The top section shows the 'Incoming' branch (7b18bdb) and the 'Current' branch (b7bd9b1). The 'Incoming' code defines `printMessage` with parameters `showUsage` and `showVersion`. The 'Current' code defines `printMessage` with parameters `showUsage` and `message`. A conflict is highlighted in the 'Result' section, where the 'Incoming' code is accepted for the `showVersion` parameter, while the 'Current' code is accepted for the `showUsage` parameter and the `message` parameter. The 'Result' code shows the merged function signature `function printMessage(showUsage) {` and the `console.log` statements. A 'Complete Merge' button is visible in the bottom right corner of the 'Result' section.

```
merge-git-playground > JS target.js > printMessage

Incoming 7b18bdb • theirs
13
14 /**
15  * Prints the welcome message
16  */
17 function printMessage(showUsage, showVersion) {
18     console.log("Welcome To Line Counter");
19     if (showVersion) {
20         console.log("Version: 1.0.0");
21     }
22     if (showUsage) {
23         console.log("Usage: node base.js <file1>");
24     }
}

Current b7bd9b1 • main
13
14 /**
15  * Prints the welcome message
16  */
17 function printMessage(showUsage, message) {
18     console.log(message);
19 }
20 if (showUsage) {
21     console.log("Usage: node base.js <file1>");
22 }

Result merge-git-playground\target.js 1 Conflict Remaining
13
14 /**
15  * Prints the welcome message
16  */
17 function printMessage(showUsage) {
18     console.log("Welcome To Line Counter");
19 }
20 if (showUsage) {
21     console.log("Usage: node base.js <file1> <file2> ...");
22 }
```



# Summary

- Repository = working tree + store
  - Store contains history
  - History is a DAG of commits
  - References, tags, and HEAD
- Commit/checkout are local operations
  - Former changes store, latter working tree
- Merge
  - Directional (merge other “into” HEAD)