

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

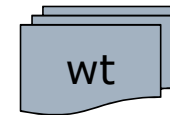
- *Repository* = working tree + store + index
 - Warning: "Repo" often used (incorrectly) to mean just the store or just the working tree
- *Working tree* = project itself
 - Ordinary directory with files & subdirectories
- *Store* = history of project
 - Hidden directory: don't touch!
- *Index* = virtual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

File Structure of a Repository

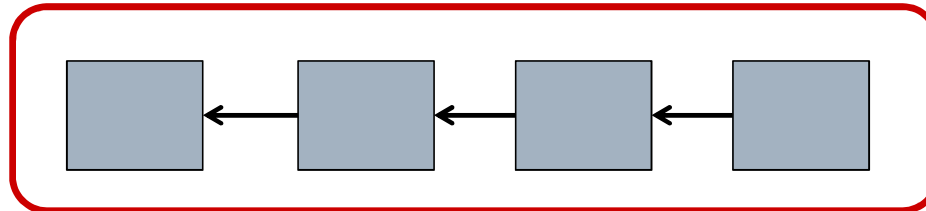
```
~/mashup/  
├── css/  
│   ├── buckeye-alert-resp.css  
│   └── demo.css  
├── demo-js.html  
├── Gemfile  
├── Gemfile.lock  
├── .git/  
│   ├── HEAD  
│   ├── index  
│   └── ...etc...  
├── .gitignore  
├── Rakefile  
├── README.md  
└── ...etc...
```

Conceptual Structure

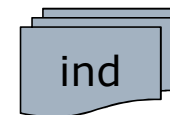
Computer Science and Engineering ■ The Ohio State University



working tree
~/mashup/

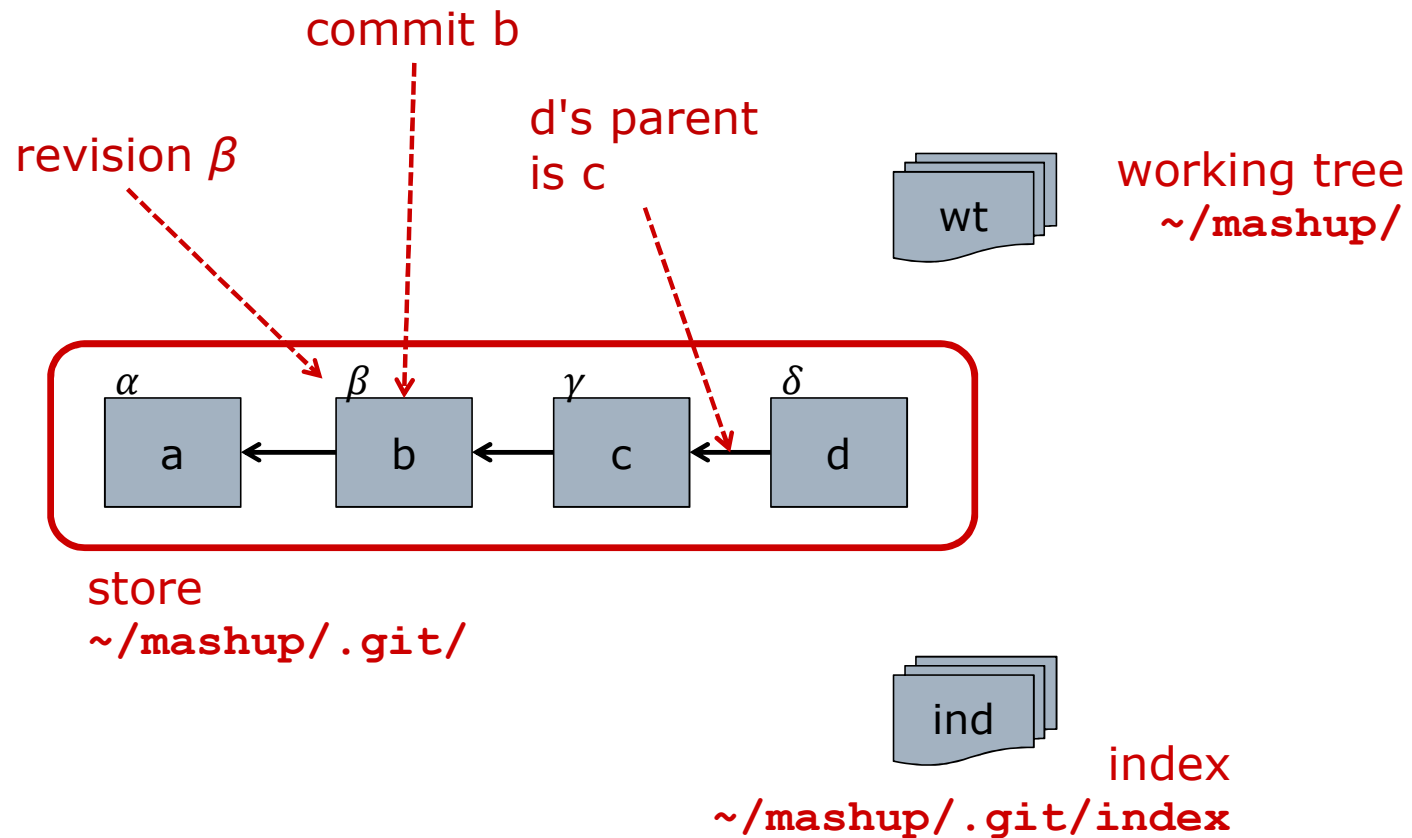


store
~/mashup/.git/



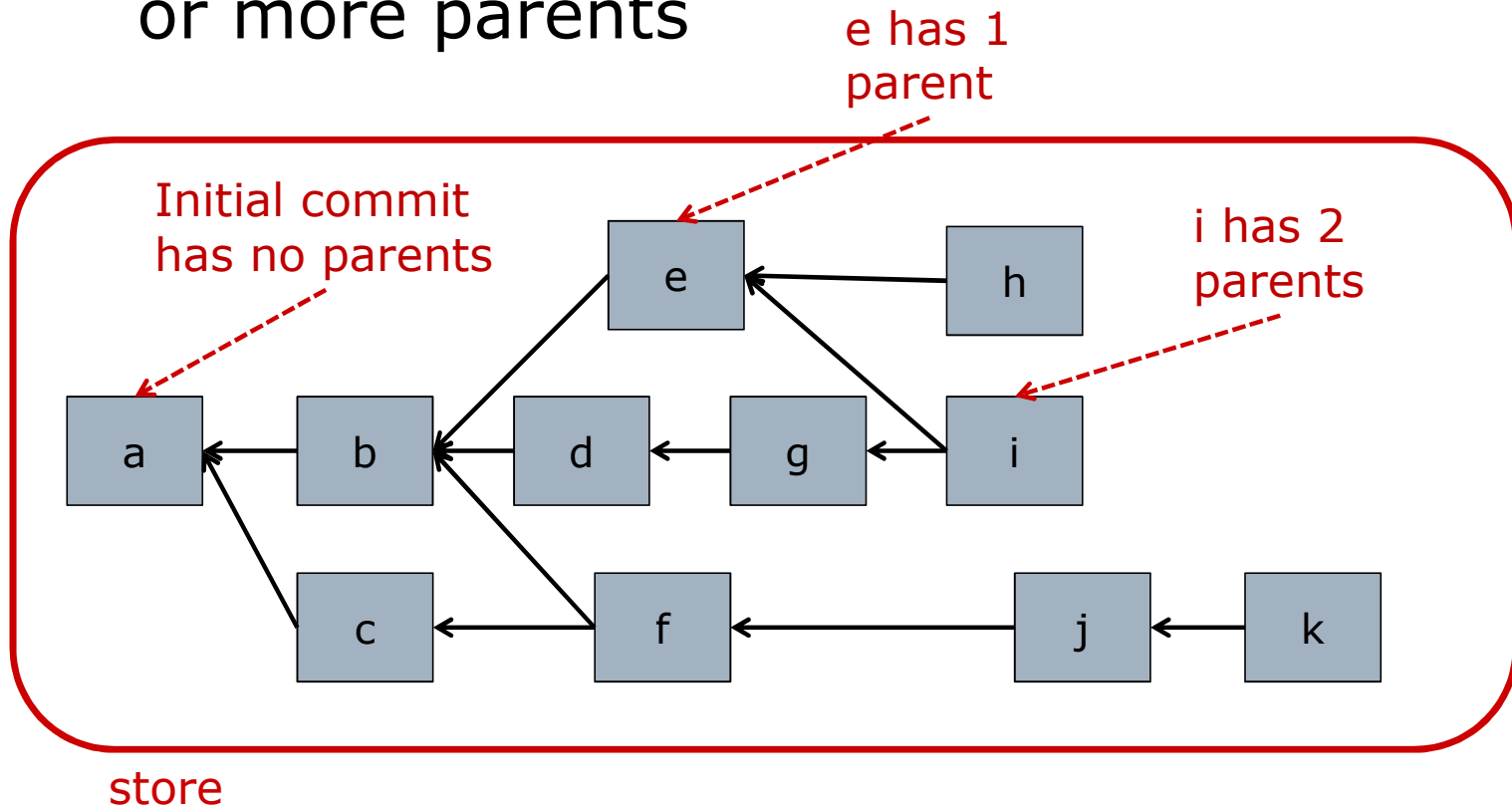
index
~/mashup/.git/index

A History of Commits

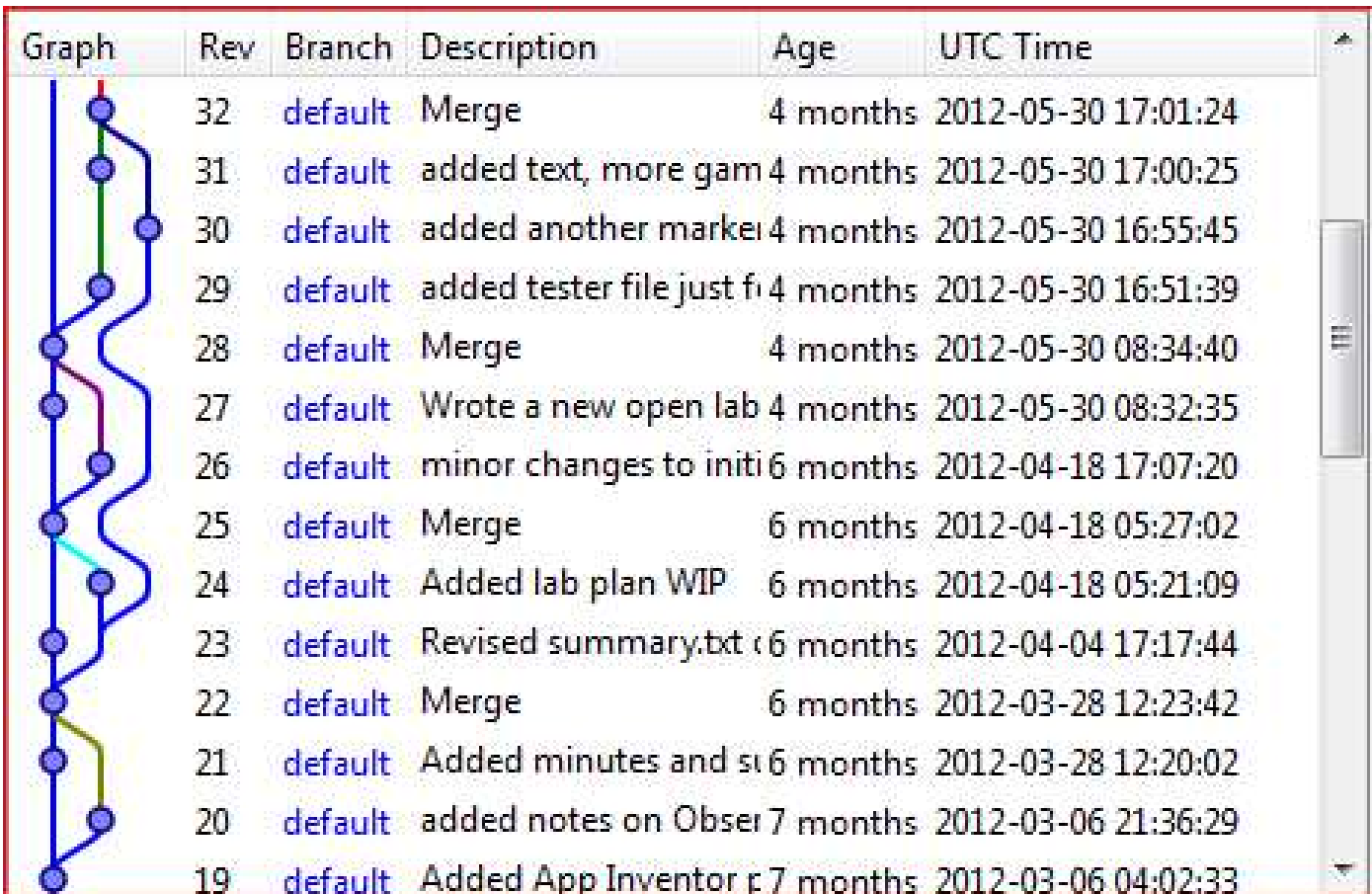


History is a DAG

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



Example View of DAG



Graph	Rev	Branch	Description	Age	UTC Time
	32	default	Merge	4 months	2012-05-30 17:01:24
	31	default	added text, more gam	4 months	2012-05-30 17:00:25
	30	default	added another marke	4 months	2012-05-30 16:55:45
	29	default	added tester file just fi	4 months	2012-05-30 16:51:39
	28	default	Merge	4 months	2012-05-30 08:34:40
	27	default	Wrote a new open lab	4 months	2012-05-30 08:32:35
	26	default	minor changes to initi	6 months	2012-04-18 17:07:20
	25	default	Merge	6 months	2012-04-18 05:27:02
	24	default	Added lab plan WIP	6 months	2012-04-18 05:21:09
	23	default	Revised summary.txt c	6 months	2012-04-04 17:17:44
	22	default	Merge	6 months	2012-03-28 12:23:42
	21	default	Added minutes and st	6 months	2012-03-28 12:20:02
	20	default	added notes on Obser	7 months	2012-03-06 21:36:29
	19	default	Added App Inventor p	7 months	2012-03-06 04:02:33

Example View of DAG

```
$ git log --oneline --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

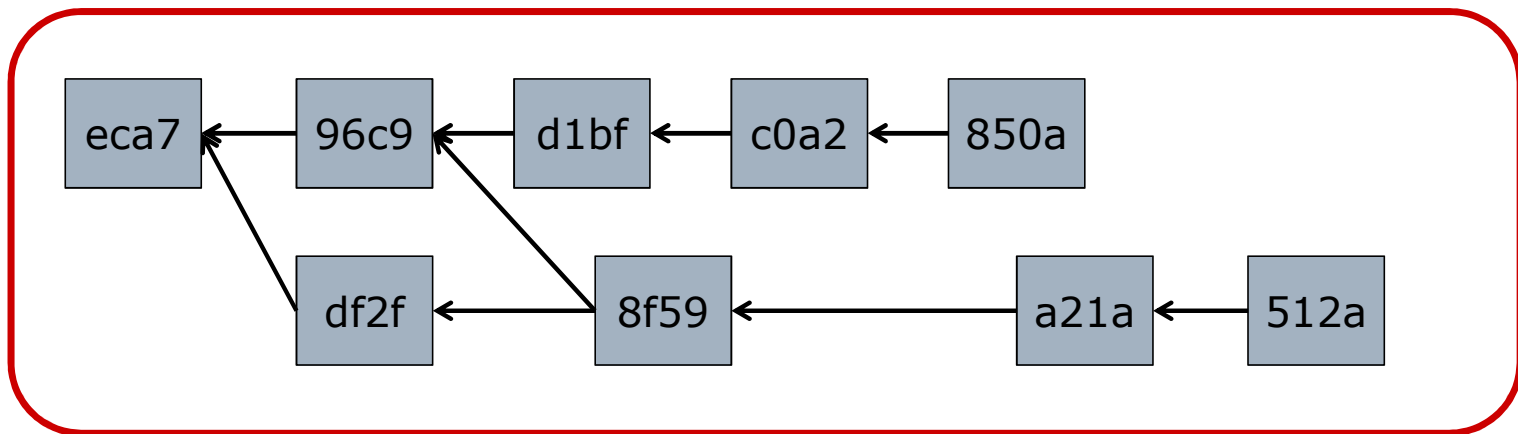
```
$ git show --name-only
16188493c252f6924baa17c9b84a4c1baaed438b
Author: Paul Sivilotti <user.pags@server.fake>
Date:   Mon Mar 31 15:30:50 2014 +0200

    clean up css

source/stylesheets/_site.css
```

History is a DAG

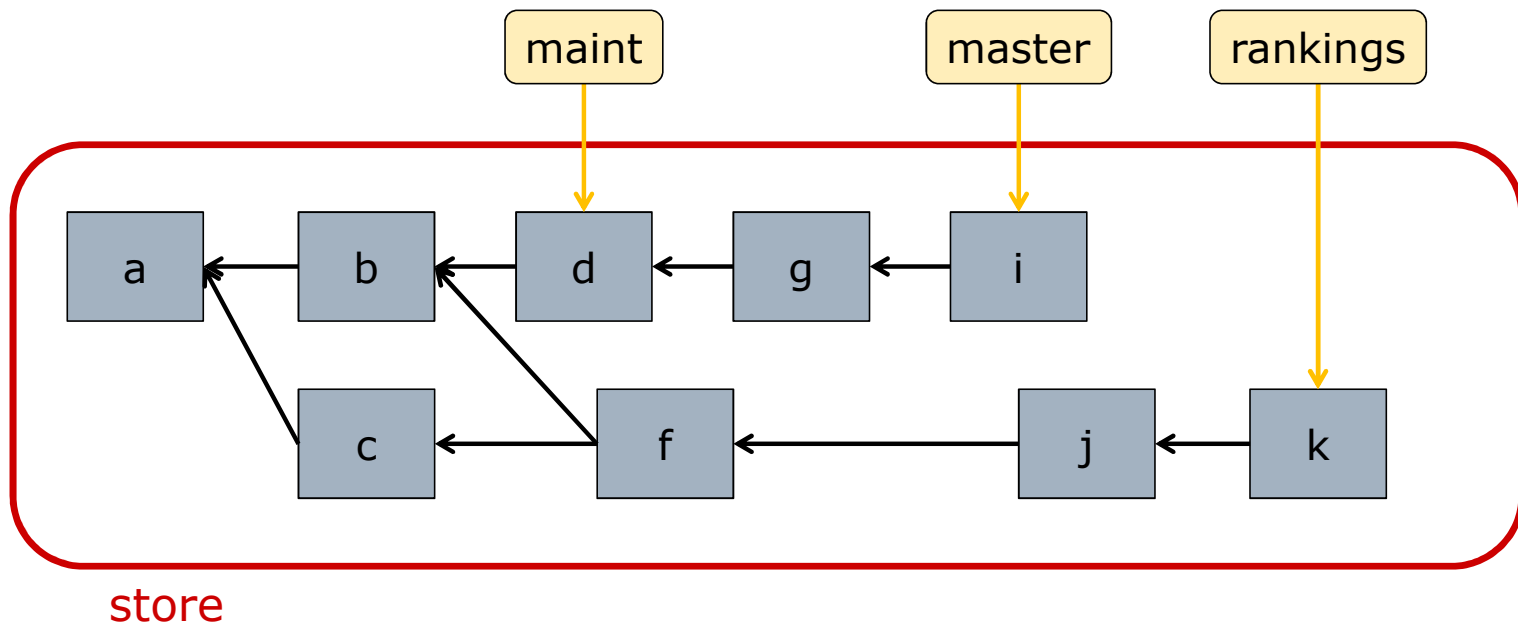
- A better picture would label each commit with its hash (prefix)



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

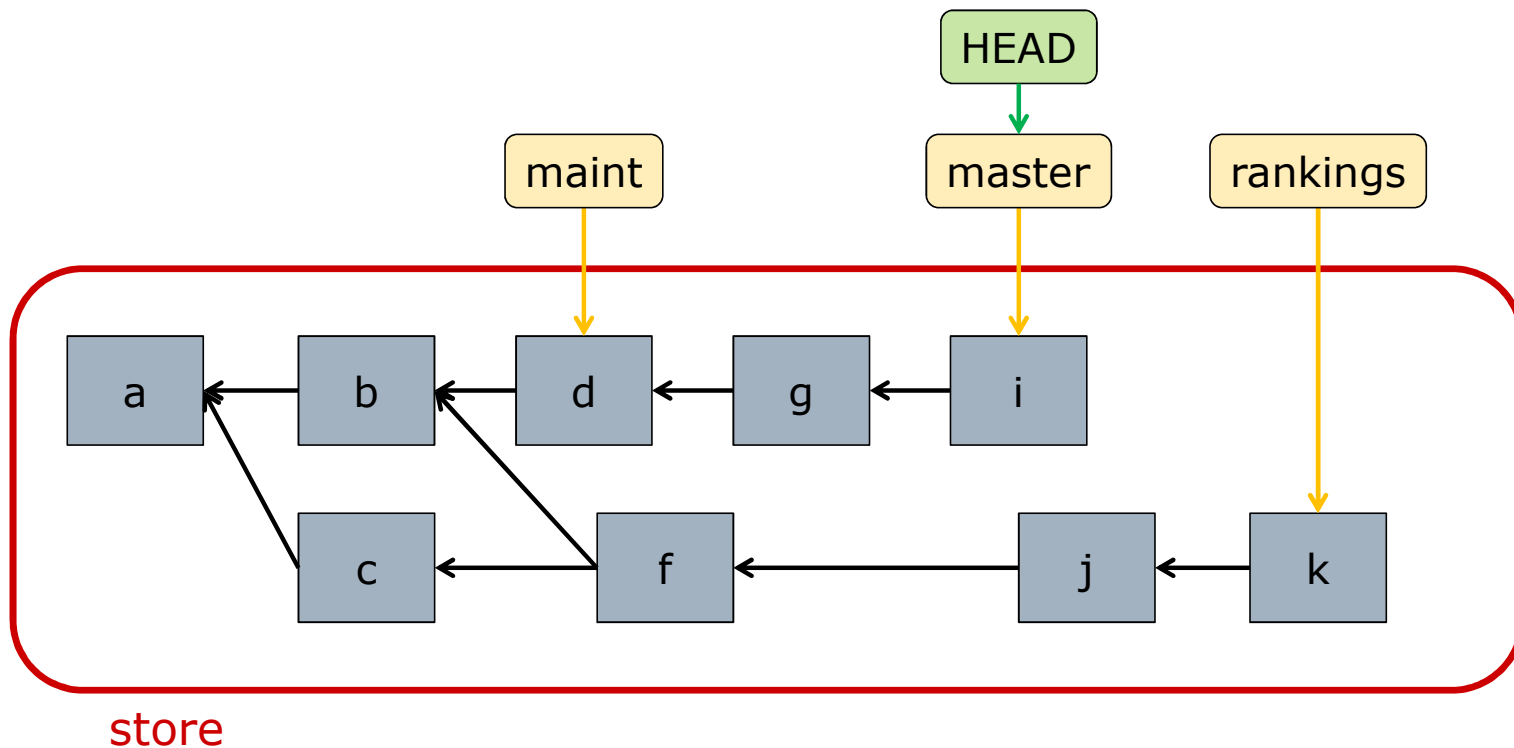
Nomenclature: Branch

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



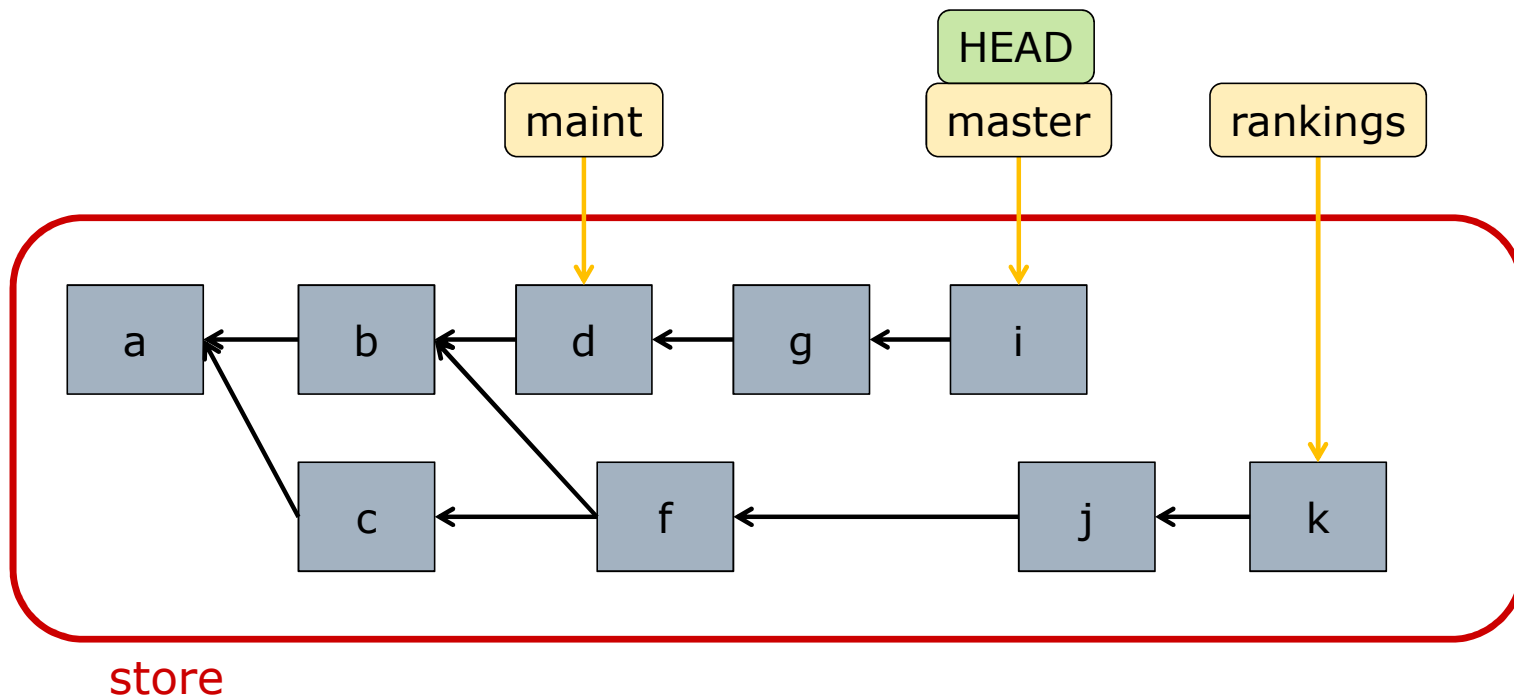
Nomenclature: HEAD

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



Nomenclature: HEAD

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

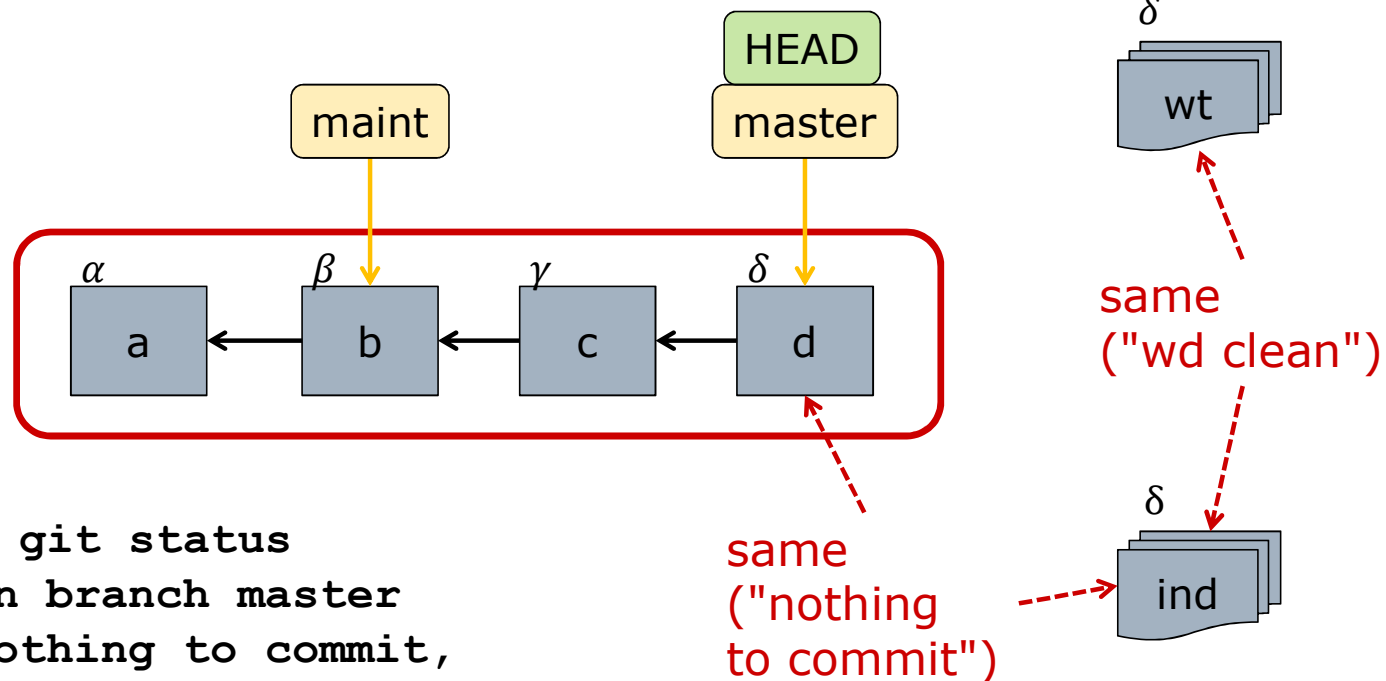


View of DAG with Branches

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

* 1618849 (HEAD -> master) 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
```

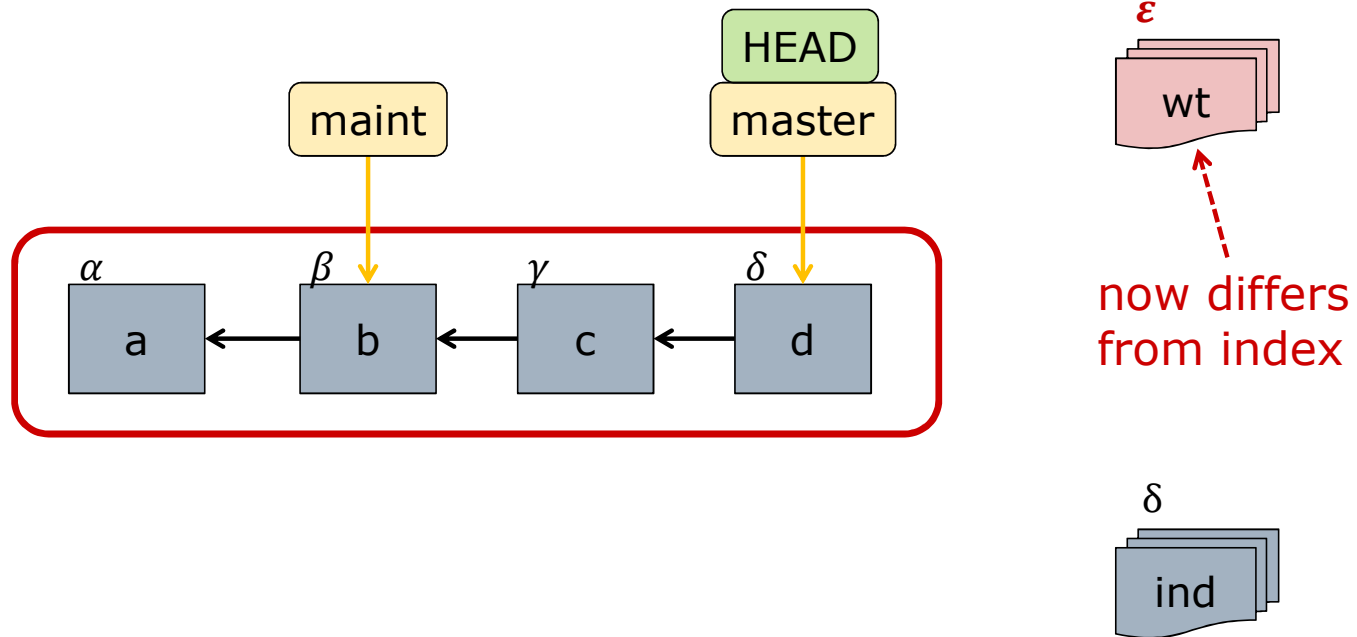
A "Clean" Repository



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

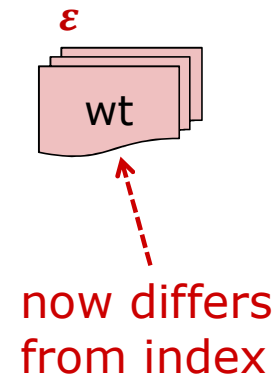
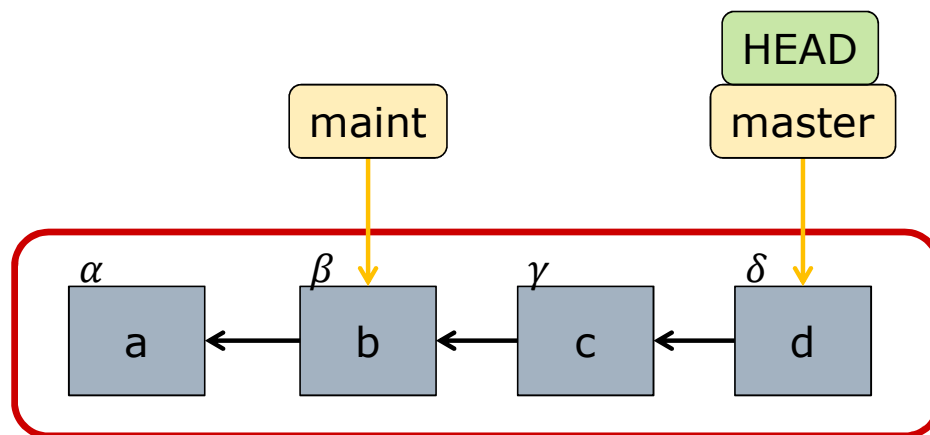

Edit Files in Working Tree

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

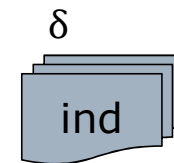


Edit Files in Working Tree

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

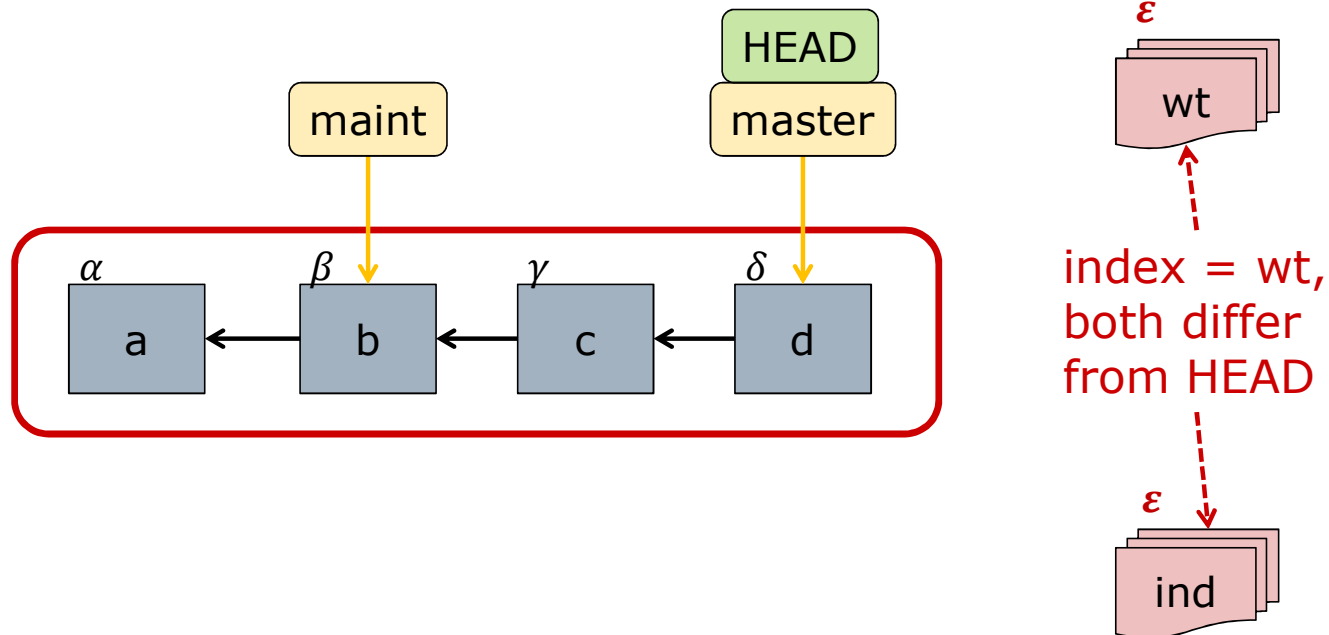


```
$ git status
On branch master
Changes not staged for commit:
  modified: css/demo.css
```



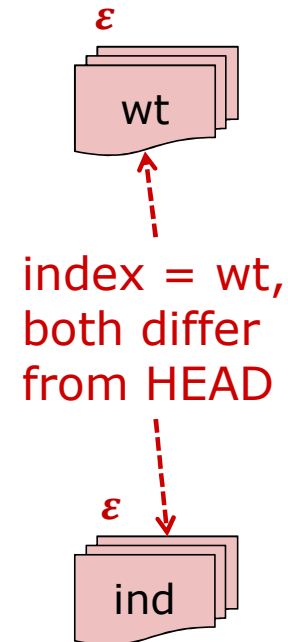
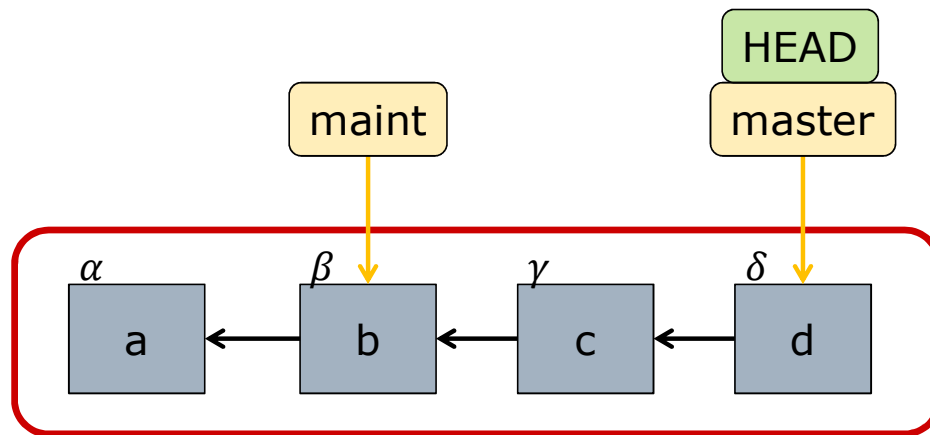
Add: Working Tree \rightarrow Index

```
$ git add --all .
```



Add: Working Tree \rightarrow Index

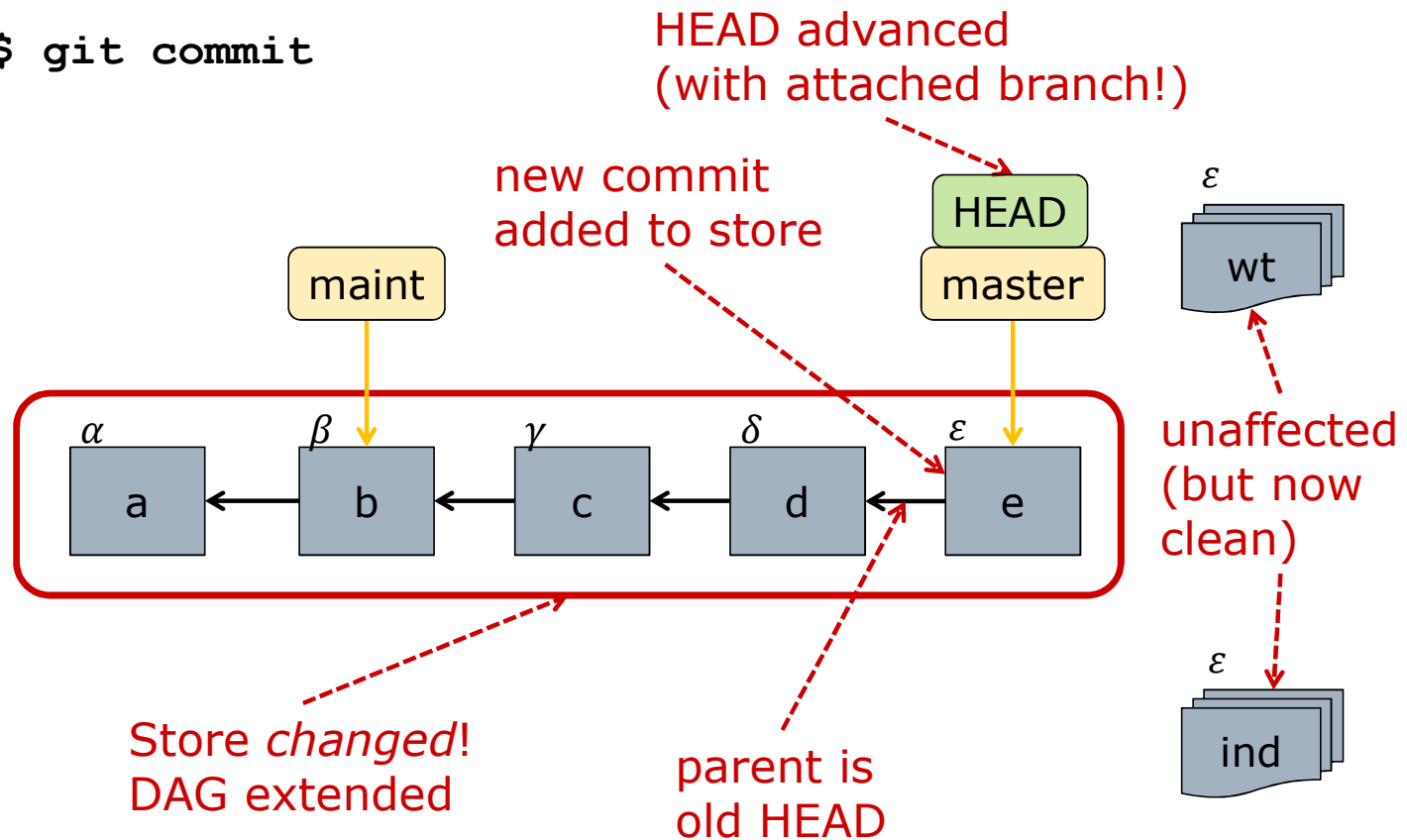
```
$ git add --all .
```



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

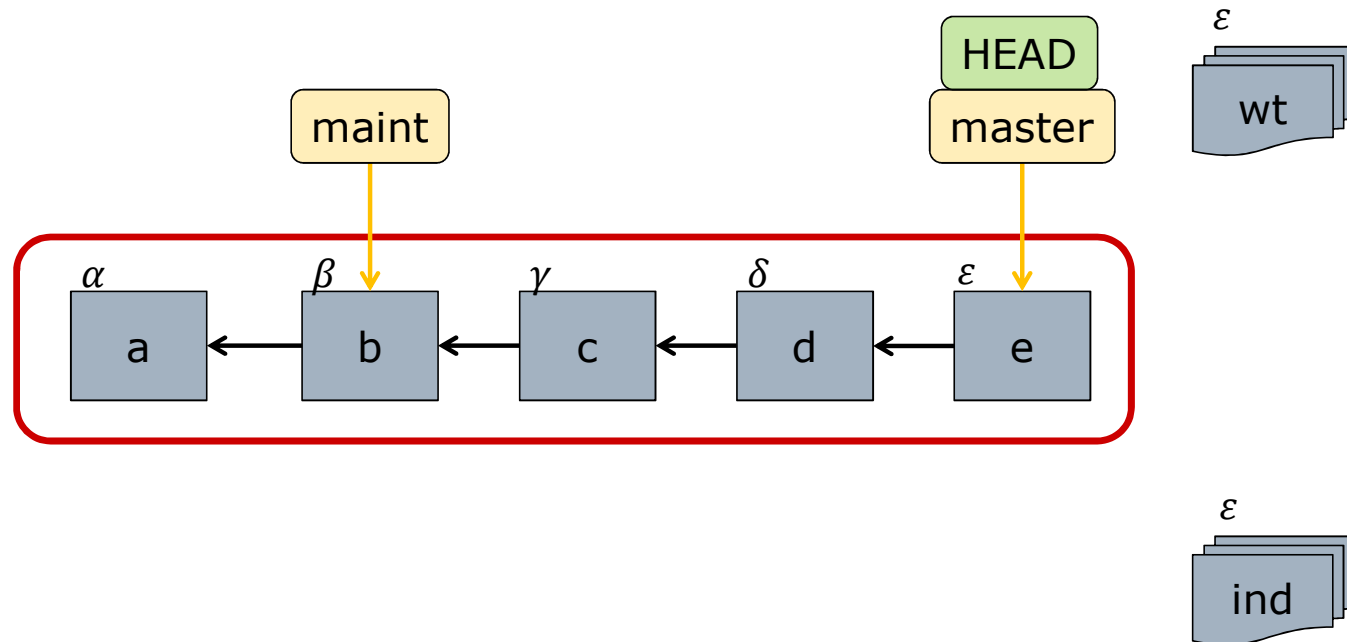
Commit: Index → Store

```
$ git commit
```



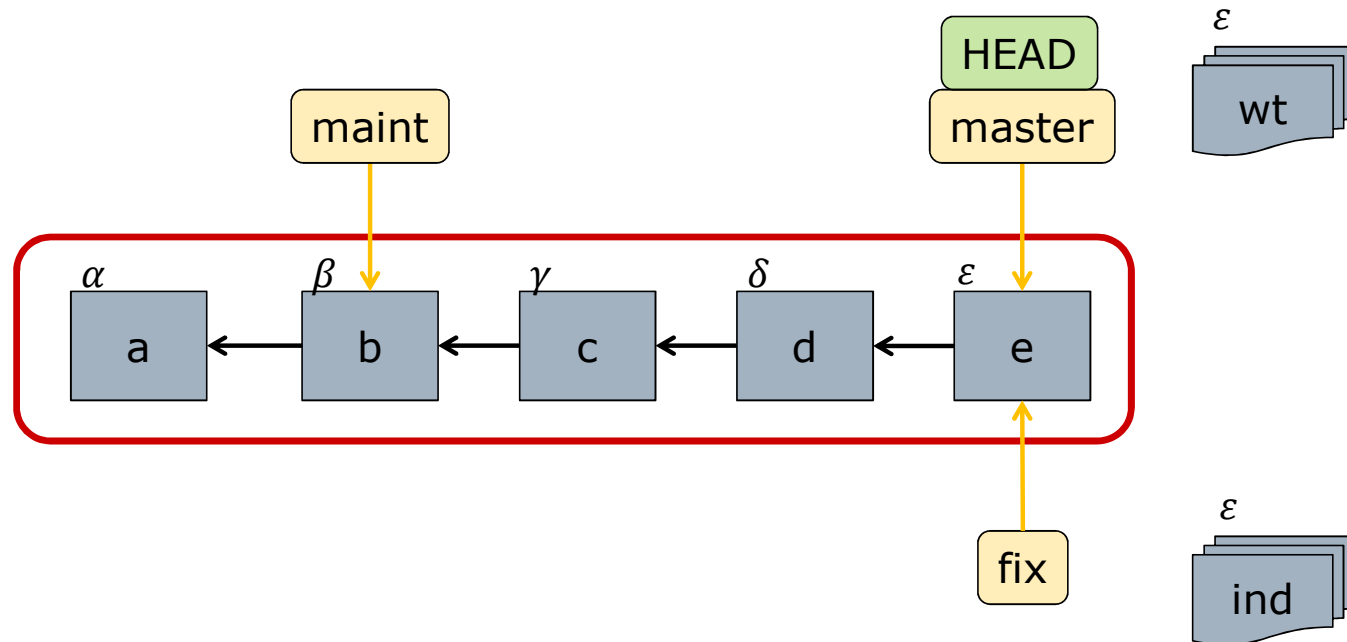
The (New) State of Repository

Computer Science and Engineering ■ The Ohio State University



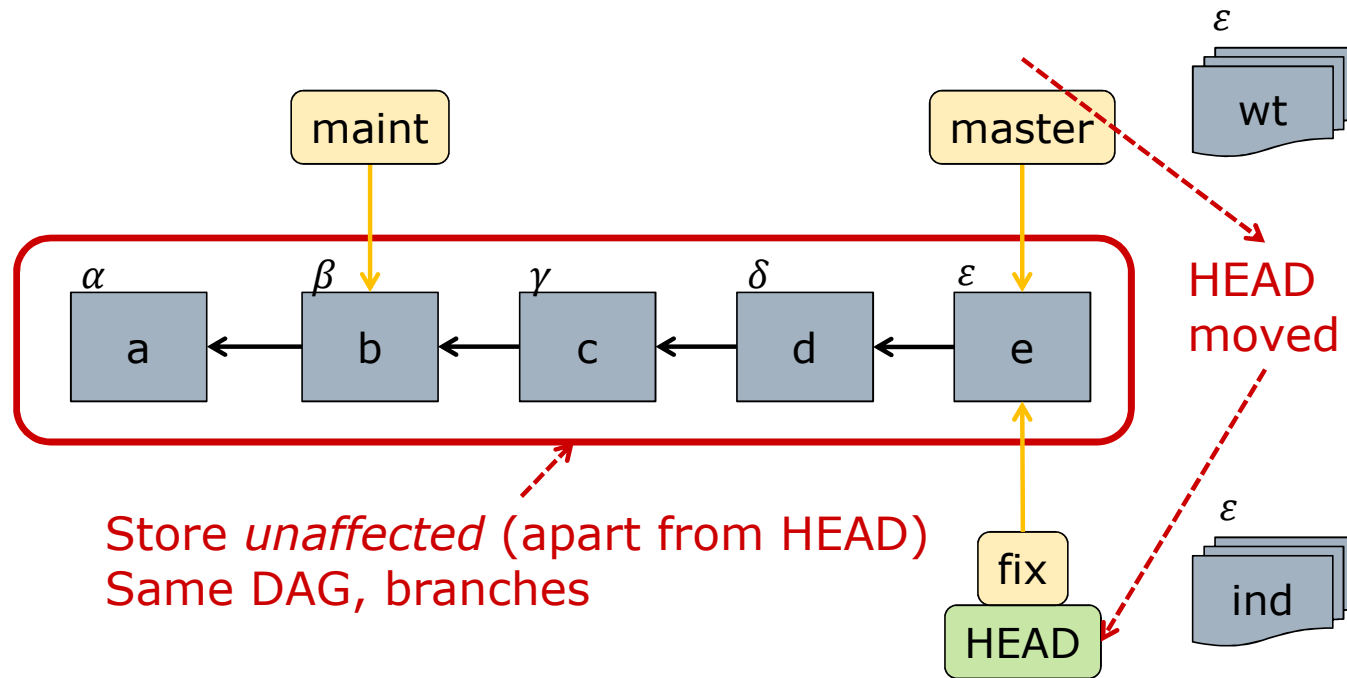
Creating a New Branch

```
$ git branch fix
```



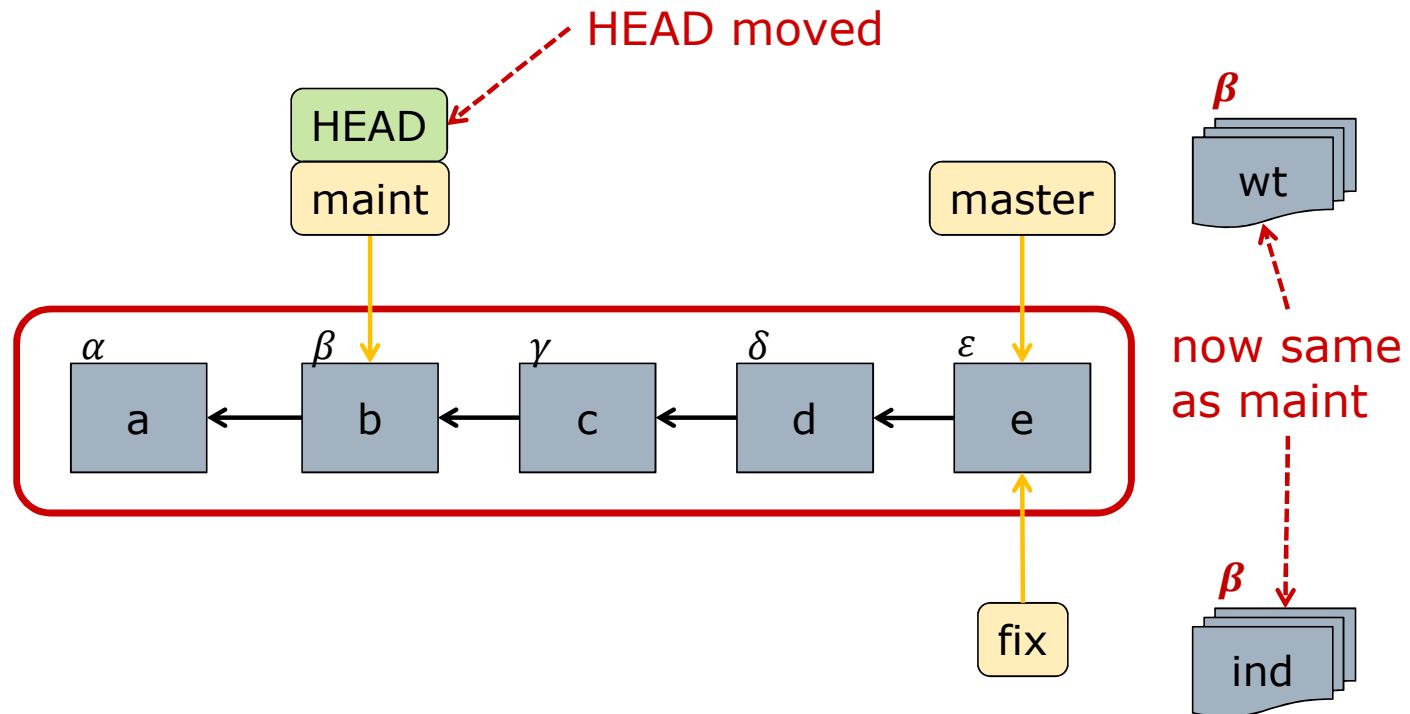
Checkout: Changing Branch

```
$ git checkout fix
```



Checkout: Changing Branch

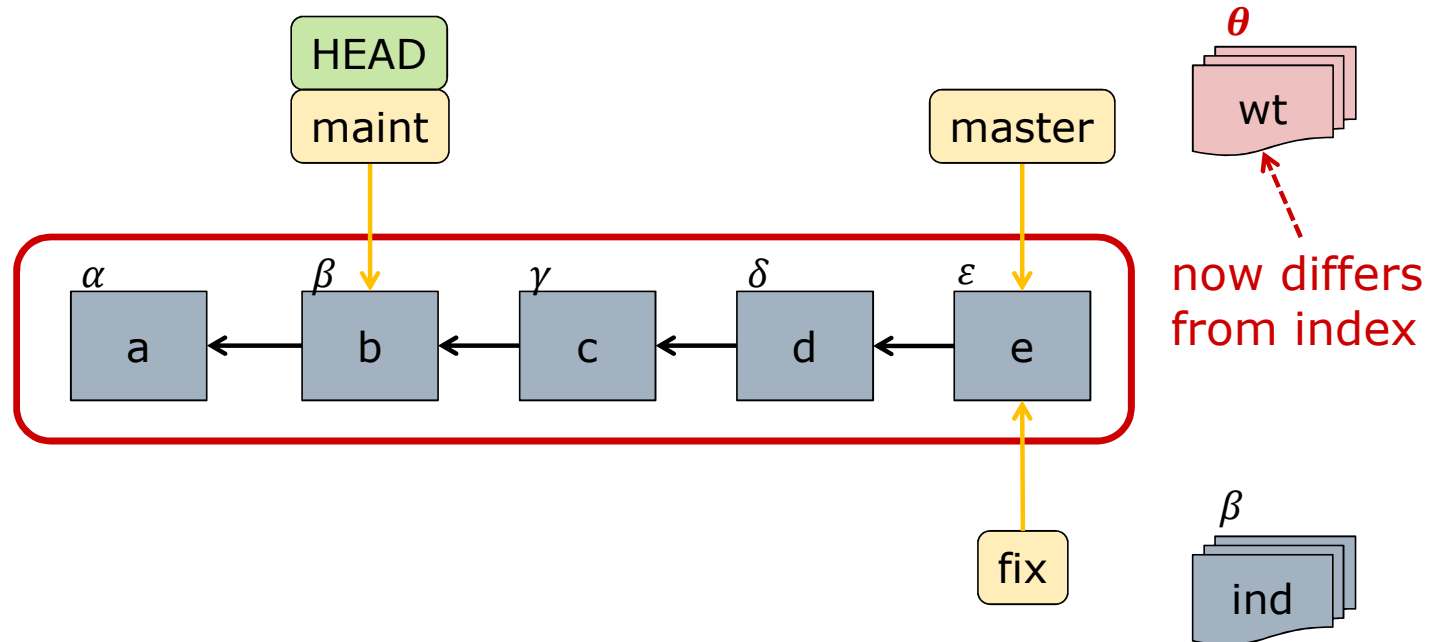
```
$ git checkout maint
```



□ Advice: checkout *only* when wt is clean

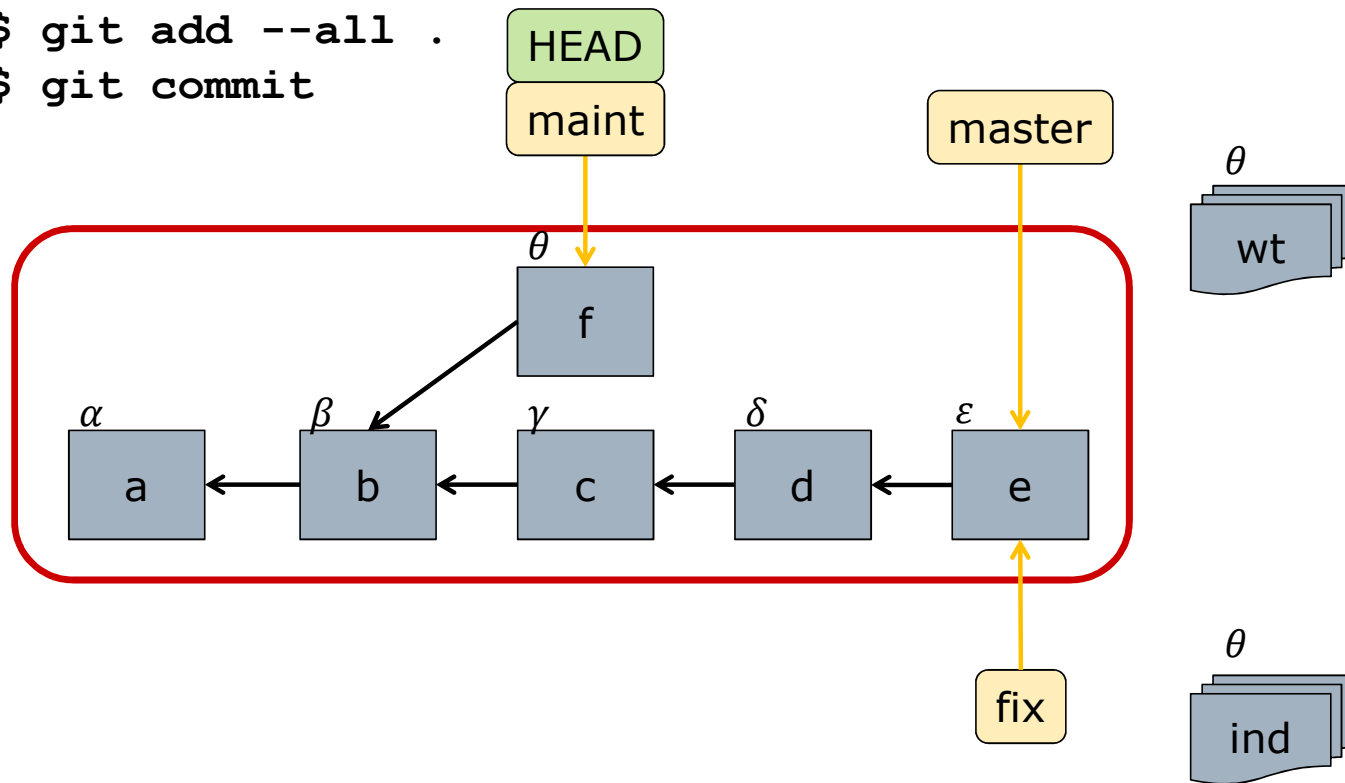
Edit Files in Working Tree

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



Add & Commit: Update Store

```
$ git add --all .  
$ git commit
```

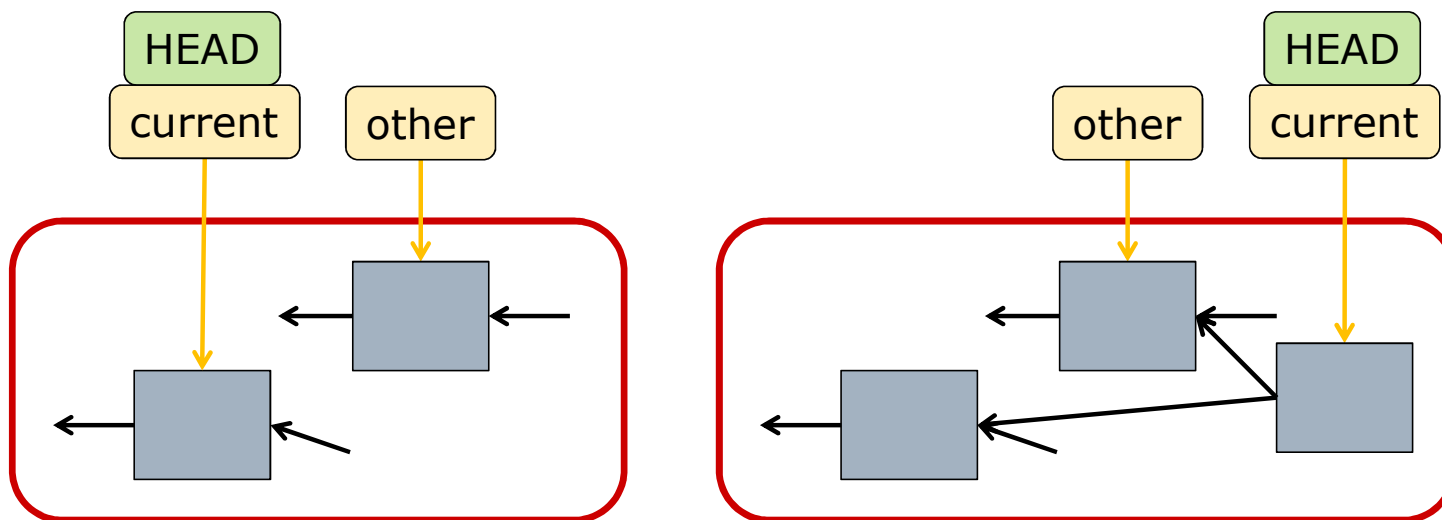


Group Formation

- Groups of 3 (or 4)
- Exchange contact information
- Each person choose a primary technical area:
 - HTML/CSS
 - JavaScript
 - Ruby
- Group constraints on choices:
 - Each technology must be represented
 - No more than 2 people per technology
- Choose a secondary interest as well
 - "Don't Care" is fine (as primary or secondary)

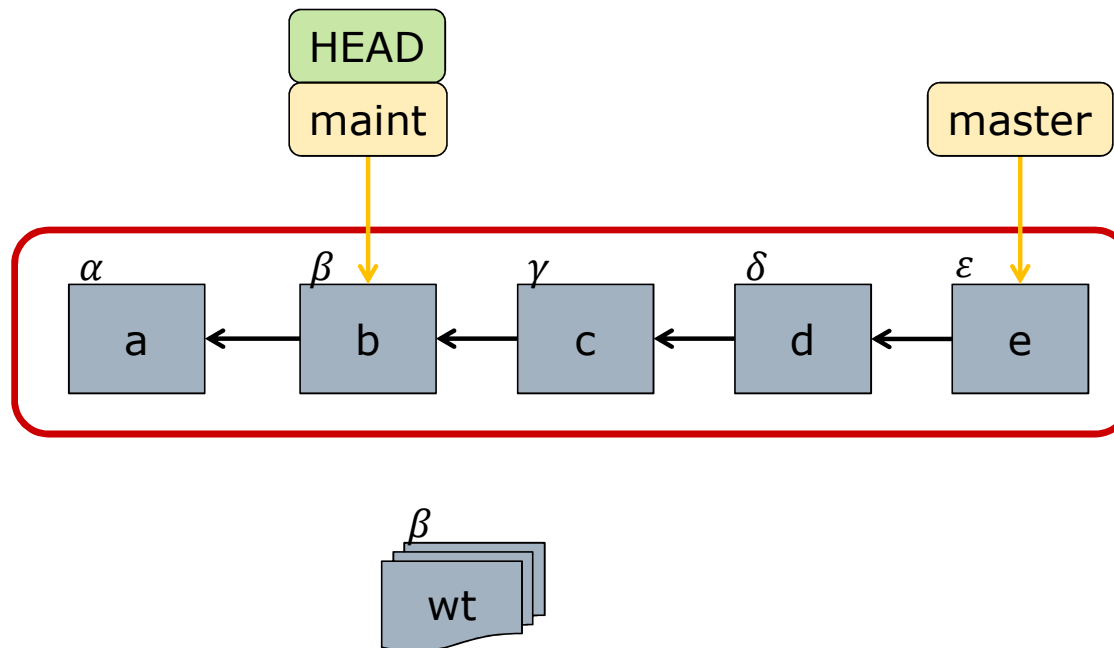
Merge: Bringing History together

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



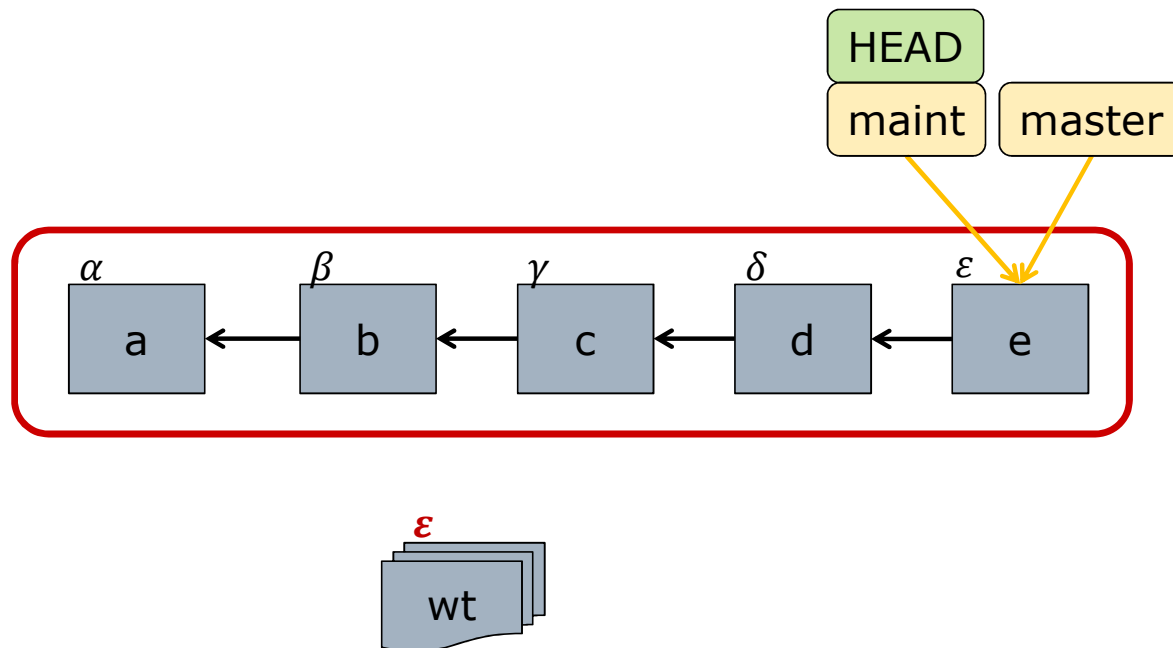
Merge – Case 1: Ancestor

- HEAD is an ancestor of other branch

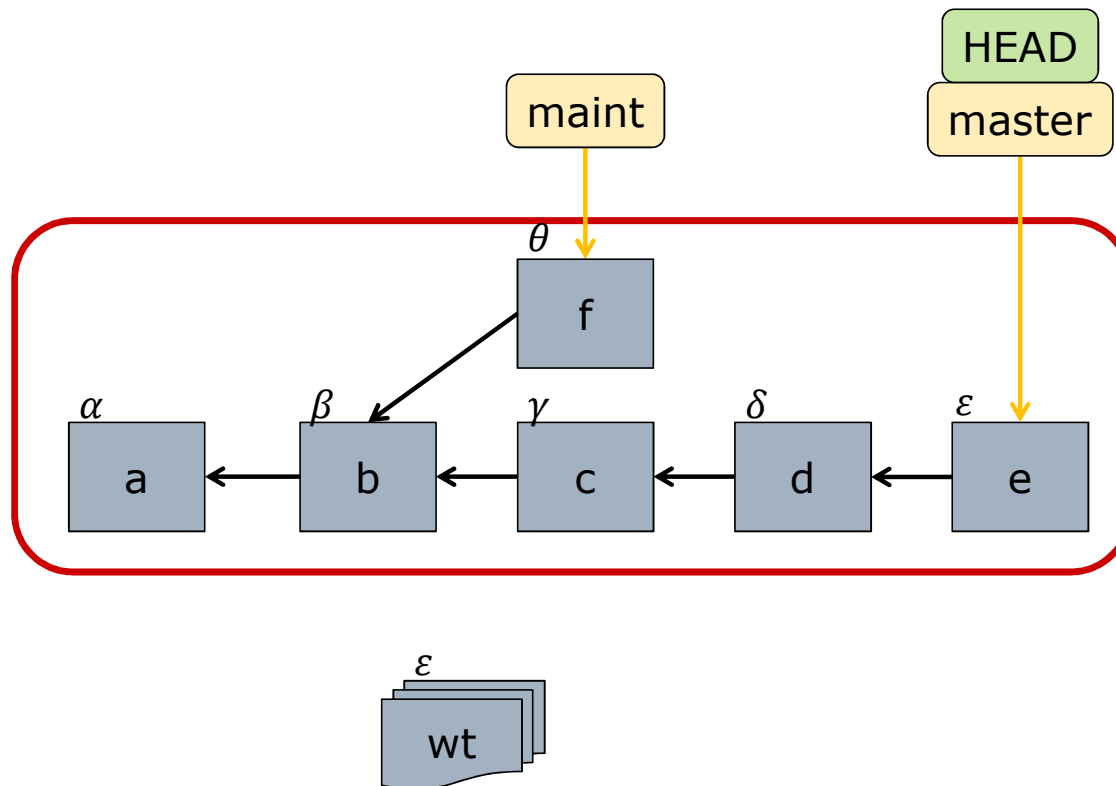


Fast-Forward Merge

```
$ git merge master
```

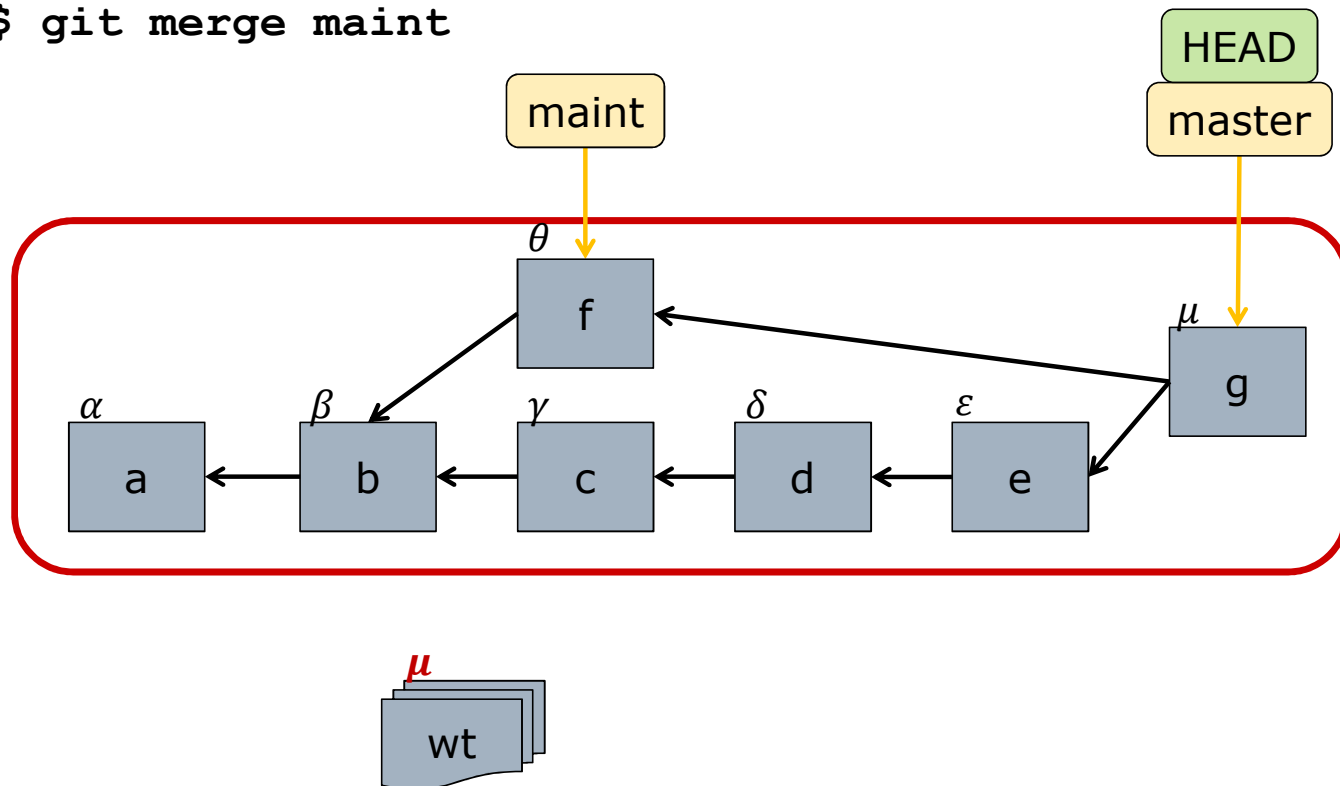


Merge – Case 2: No Conflicts



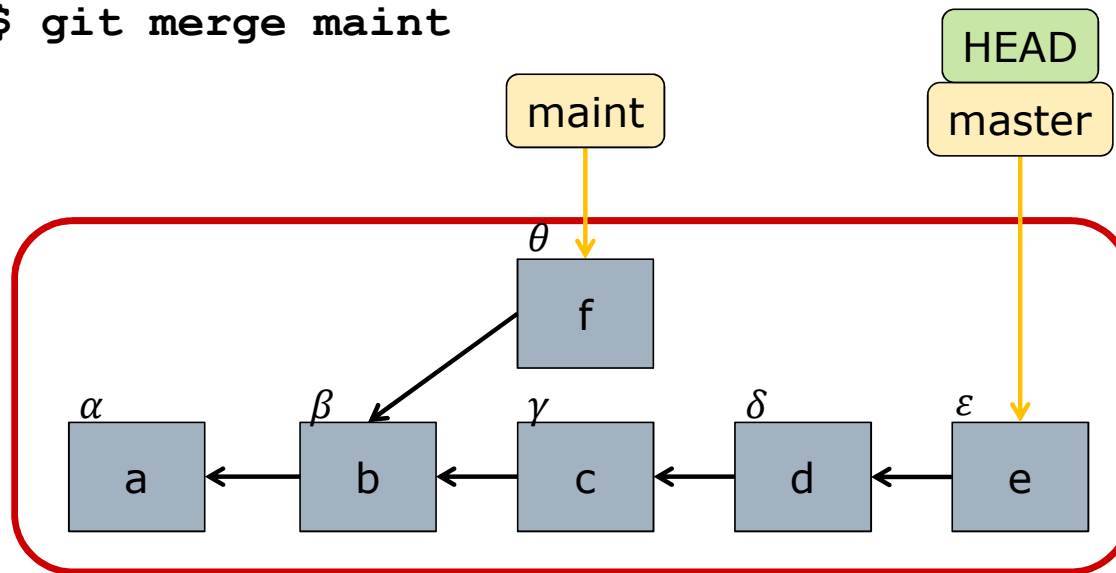
Merge Automatically Commits

```
$ git merge maint
```

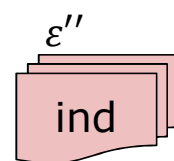
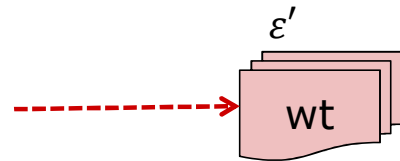


Merge – Case 3: Conflicts Exist

```
$ git merge maint
```



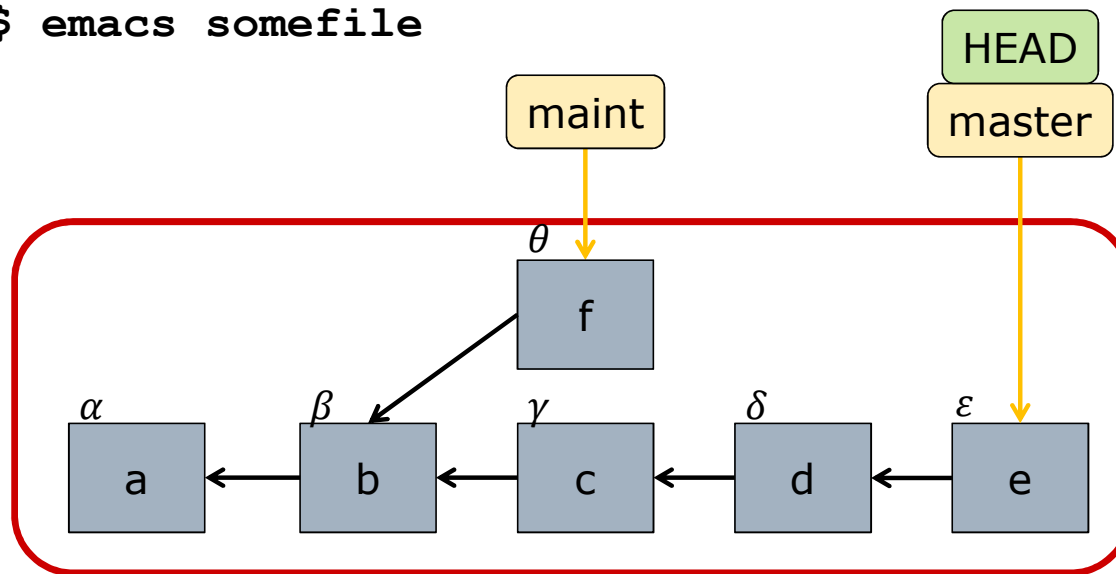
files with
conflicts
marked



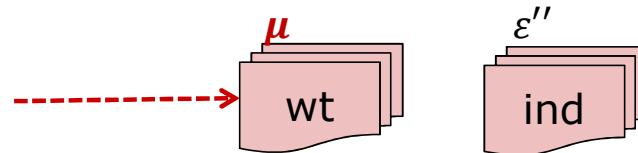
files that could
be merged
automatically

Merge: Resolve Conflicts

```
$ emacs somefile
```

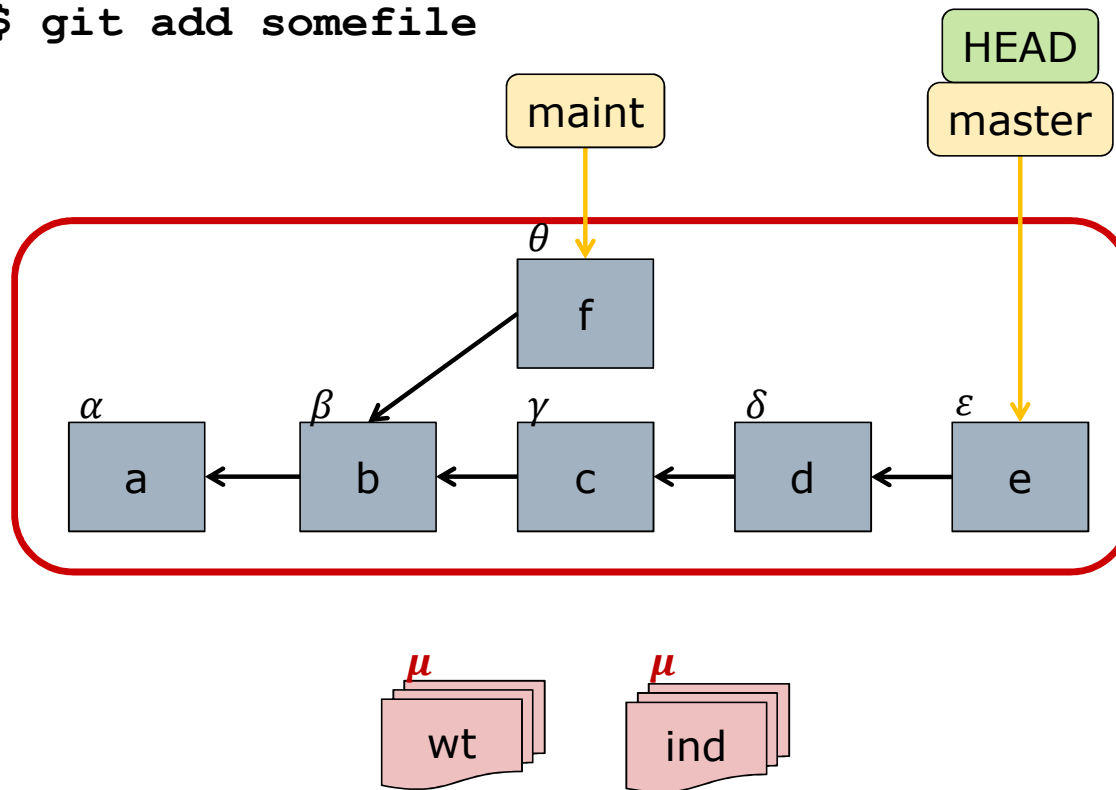


files with
conflicts
resolved



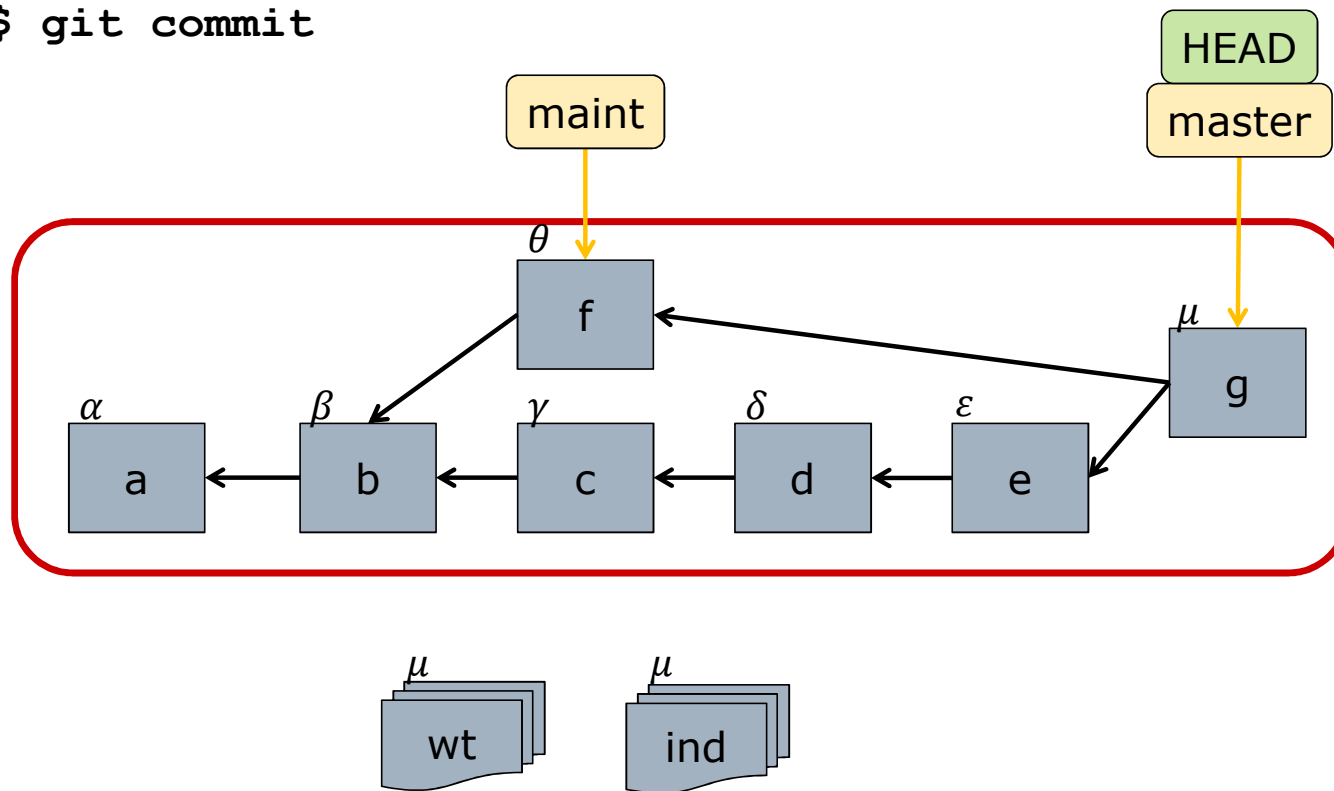
Merge with Conflicts: Add

```
$ git add somefile
```



Merge with Conflicts: Commit

```
$ git commit
```



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 A "into" B)