git

git clone basics

Hackathonners

version control

what is this?

Hackathonners

version control

- **records changes to a file (or set of files)**
- **allows to recall specific versions later**
- **allows to revert files back to a previous state**
    - even the entire project can be reverted
- **provides comparison between distinct states**
- **registers author or authors of the changes**

version control — types

- **local**
    - copy files into another directory
    - *rcs* — keeps patch sets in a special format on disk
- **centralized**
    - single server contains all versioned files — single point of failure
    - *svn, perforce, cvs*
- **distributed**
    - clients fully mirror entire repository
    - even when all servers fail, repository can be recovered from client's copy
    - allows collaboration with different groups of people
    - *git, mercurial, bazaar*

about git

a short history

Hackathonners

about git

- **started in 2005, by Linux development community**
    - In particular, Linus Torvalds, the creator of Linux
    - Linux kernel project began using a proprietary DVCS called BitKeeper
- **developed with the following goals:**
    - speed
    - simple design
    - strong support for non-linear development
    - fully distributed
    - able to handle large projects like the Linux kernel

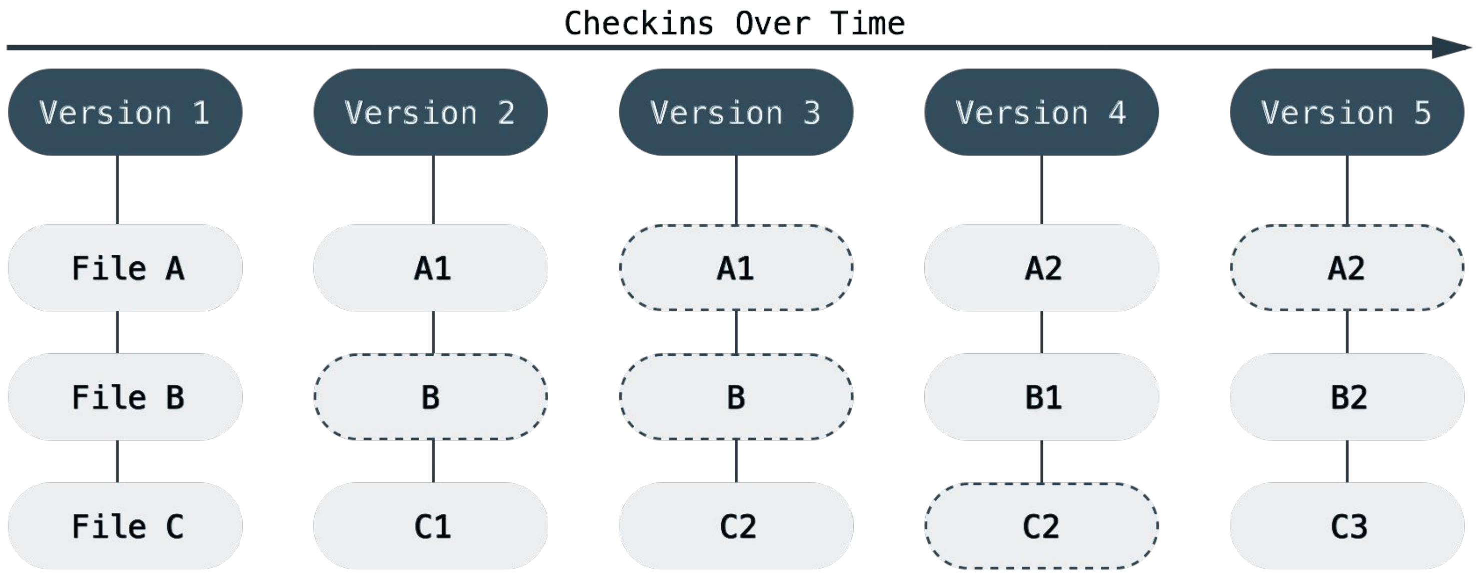
git

in a nutshell

Hackathonners

git — what is

- **is like a mini filesystem or a stream of snapshots**
  - everytime state is changed and saved, it takes a picture (snapshot) of current files
  - in new snapshots, it stores a link for previous files that have not changed



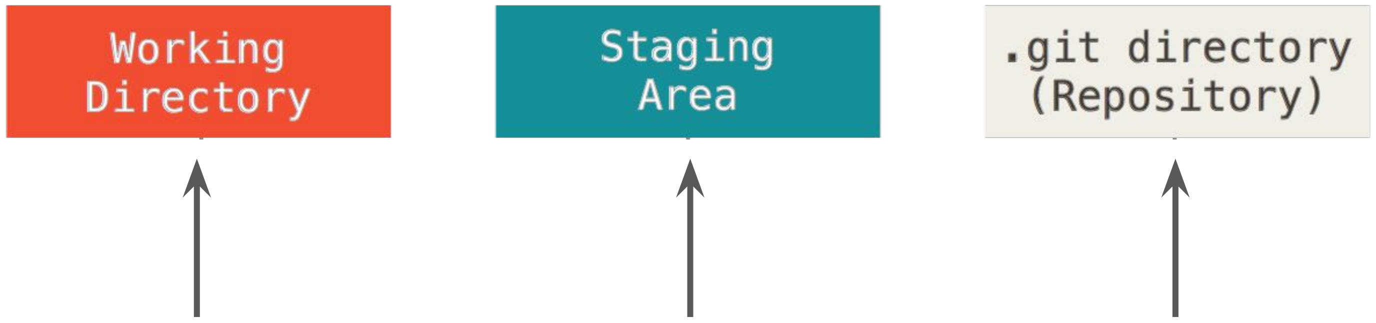
git — what is

- **almost all operations are performed locally**
  - no additional information is needed from another computer
  - history is stored in the local “database”
  - it calculates locally the difference of multiple versions
  - every change in files is locally stored
- **all changes are known**
  - integrity is one of the most important characteristics
  - it is impossible to change a file without git knowing about it
- **only adds data**
  - it is very hard to erase data in any way

git — three states of files

- **commited**
  - data is safely stored in local database
  - ***each commit is identified by a hash code (4c1f6...)***
- **modified**
  - file has changed but those changes are not commited to database yet
- **staged**
  - marked one or more files to go into next commit

git — three sections of a project



Single checkout of one version of project.

pulled from .git database and placed on disk to use or modify

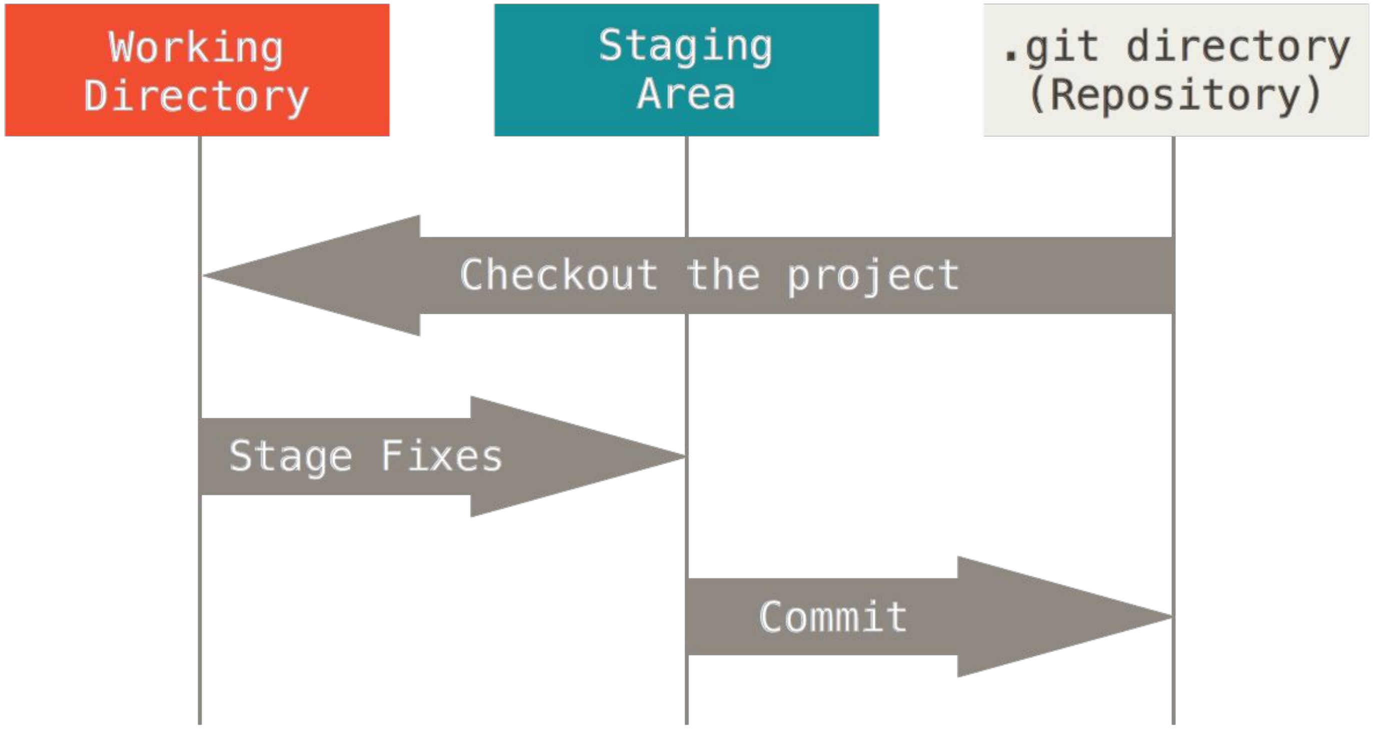
Information about what will go into the next commit

a single file stores it

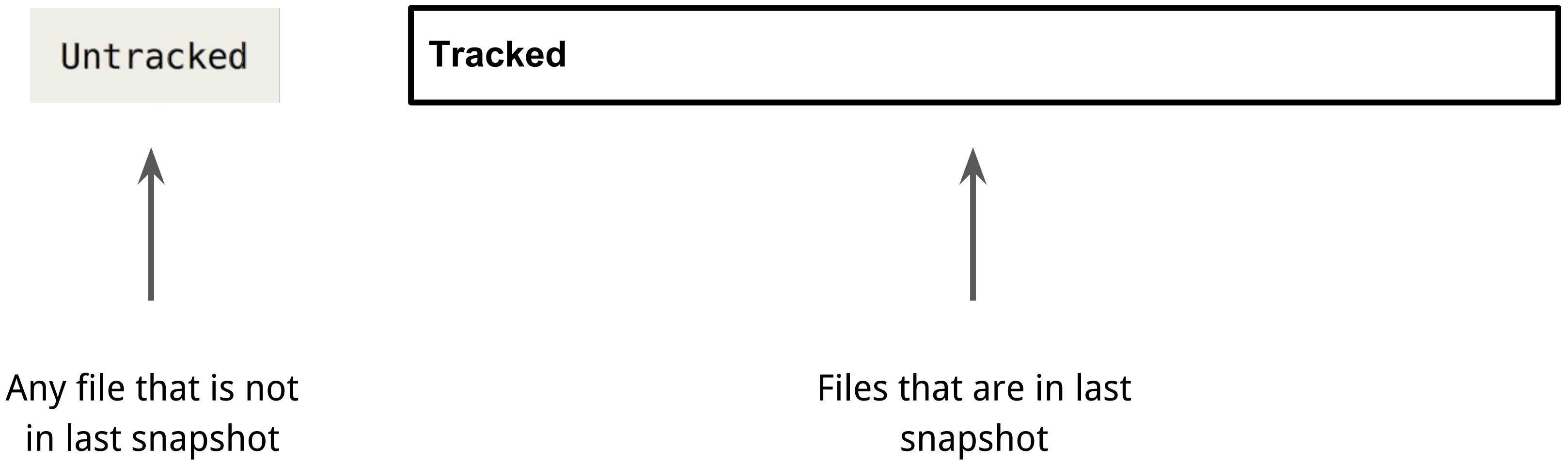
Metadata and object database for project

most important part of a git project

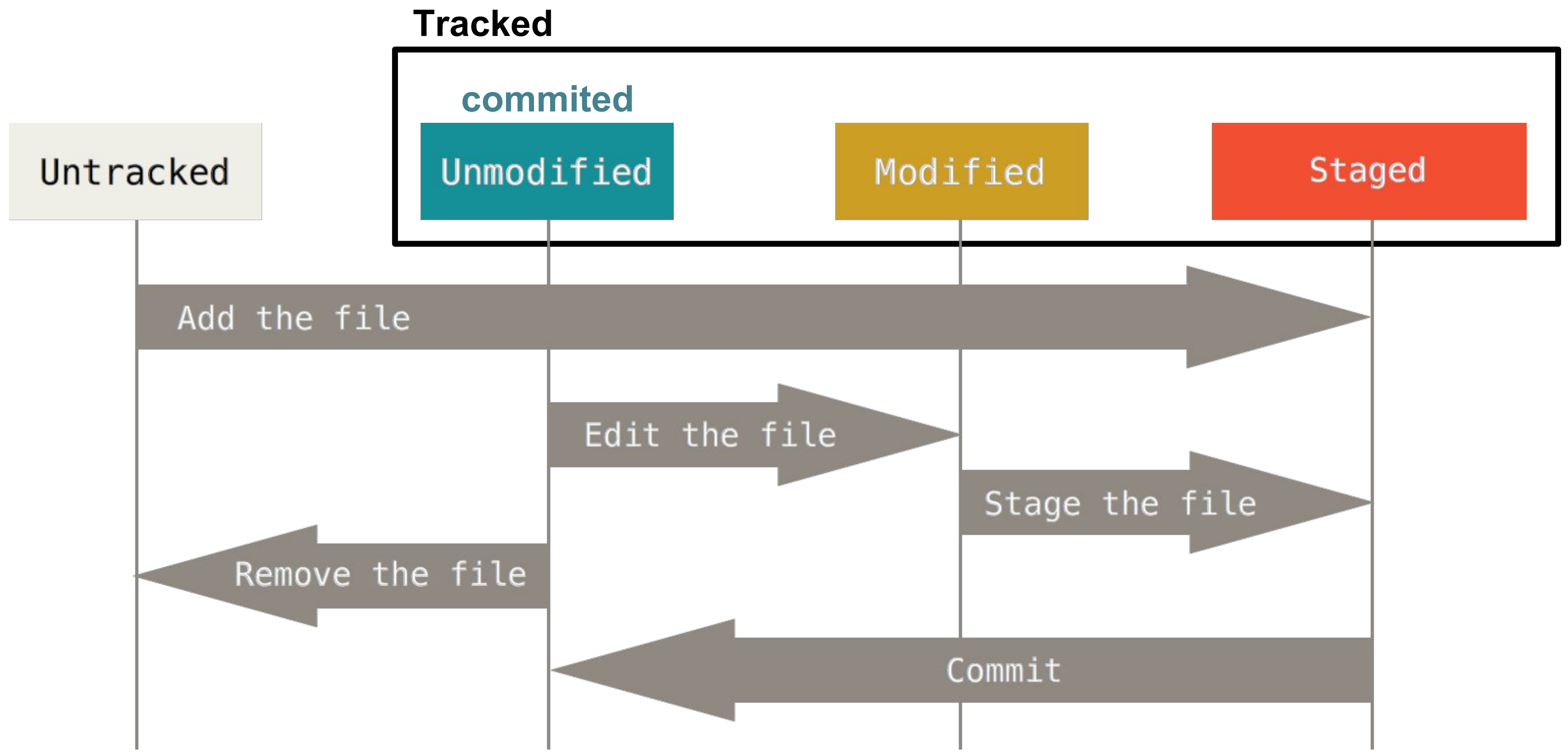
git — three sections of a project



git — lifecycle of the status of files


Untracked
Tracked
Any file that is not in last snapshot
Files that are in last snapshot

git — lifecycle of the status of files



git

in practice

Hackathonners

git — installation

git-scm.com/downloads

git — create and clone

- **init —** create an empty repository
  - `$ git init <folder>`
- **clone —** copy a remote repository to local disk
  - `$ git clone <url>`
  - `$ git clone <path>`

git — status

- **determines which files are in which state**
  - `$ git status`

git — tracking and changing files

- start tracking new files
    - `$ git add <file>`
    - `$ git add <folder>`
- stop tracking files
    - `$ git rm <file>`
    - `$ git rm <folder>`
- unstage files
    - `$ git reset HEAD <file>`
- discard changes in a given file
    - `$ git checkout -- <file>`

git — commit

- save modified (that are staged) files
    - `$ git commit -m <message>`

git — log

- **displays the log of repository**
  - `$ git log`
  - `$ git log -p <n>`
    - `shows differences in <n> commits`
  - `$ git log --stat`
    - `shows log with statistics`
  - `$ git log --graph`
    - `shows log as a graph`

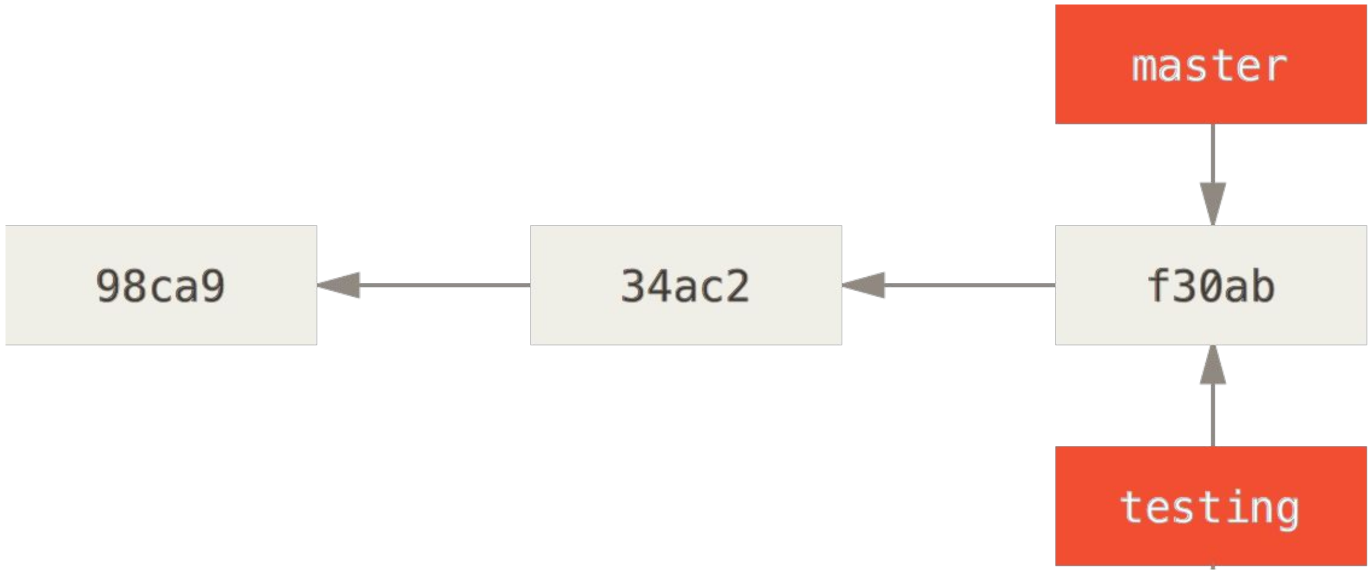
git branching

in a nutshell

Hackathonners

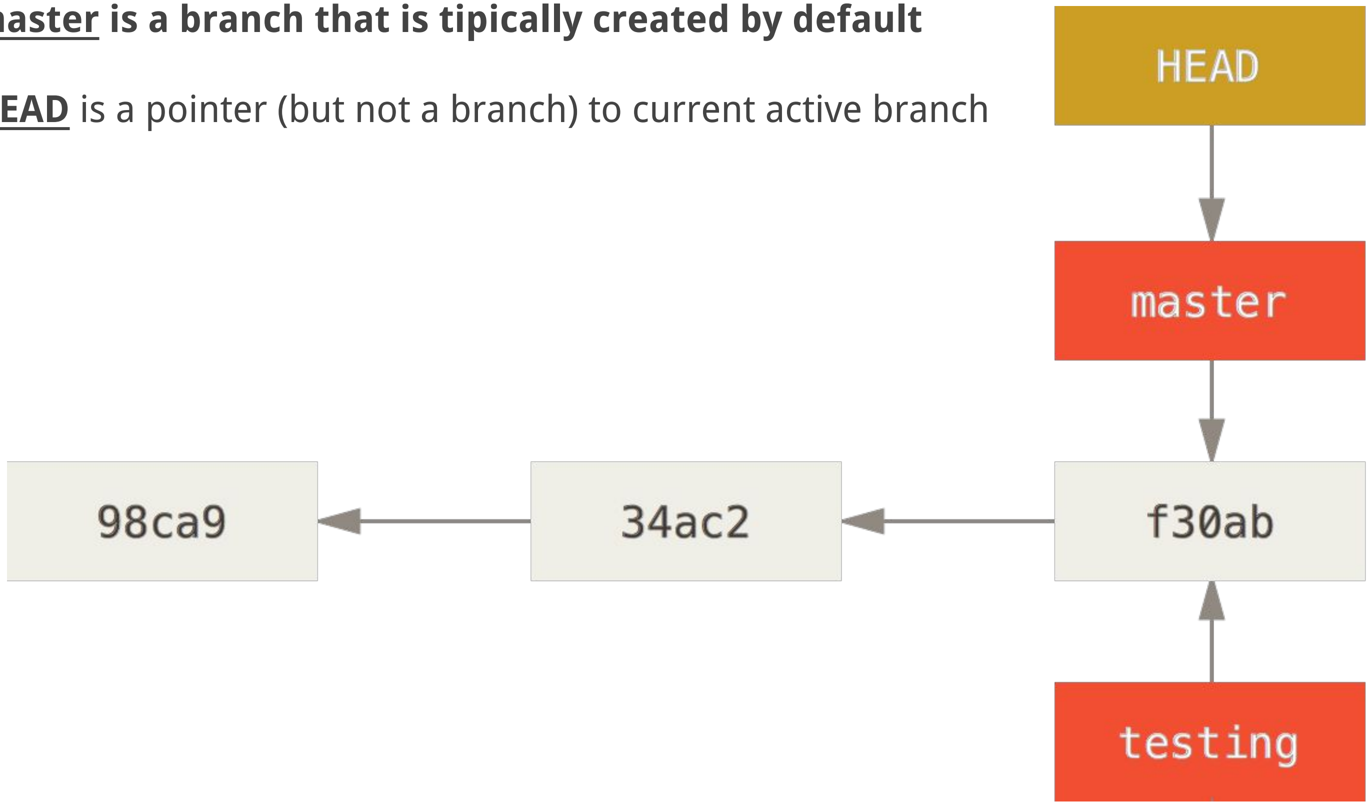
git branch — what is

- is like a new pointer to the current commit
  - master is a branch that is tipically created by default



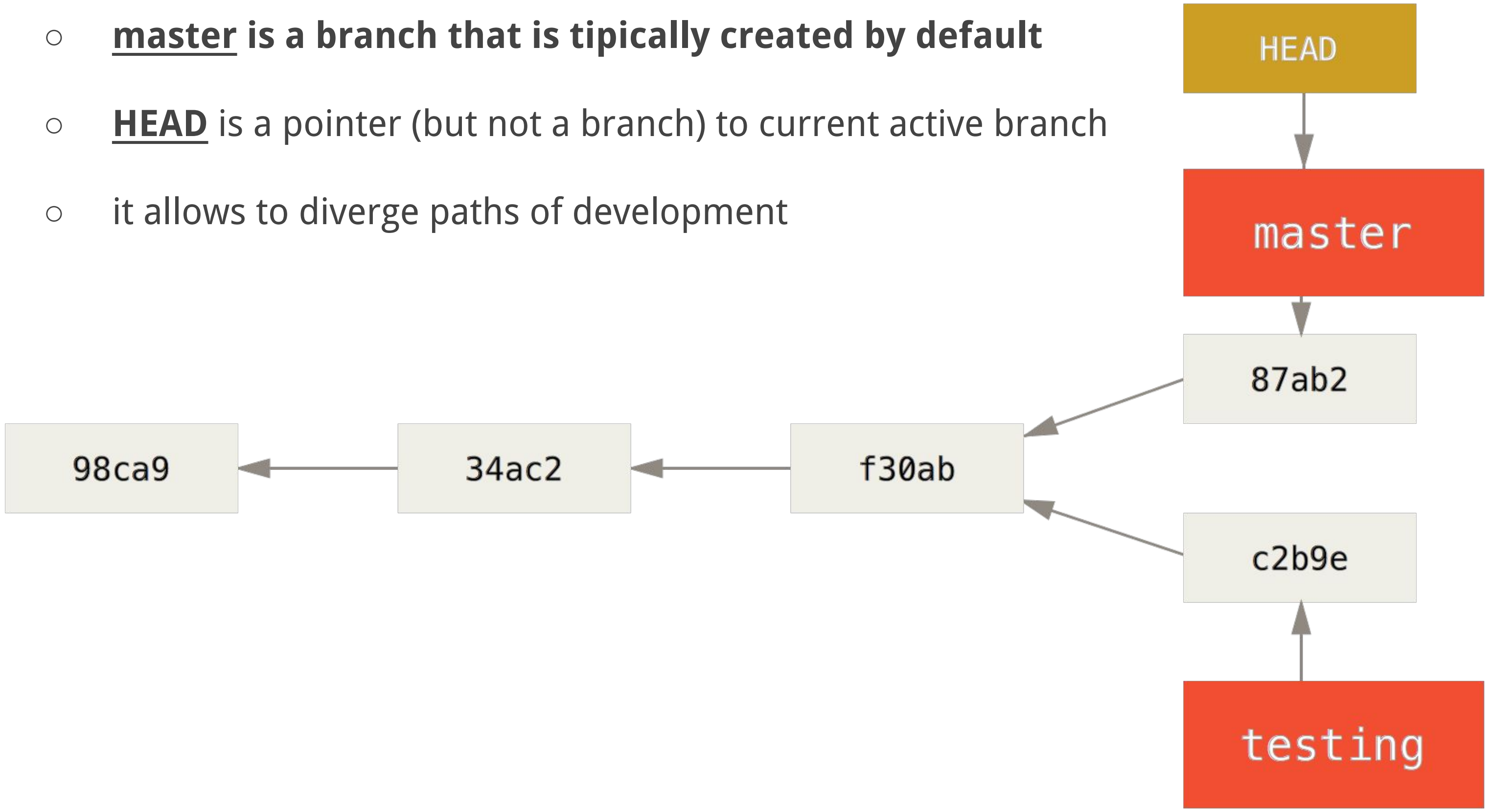
git branch — what is

- **is like a new pointer to the current commit**
    - **master is a branch that is tipically created by default**
    - **HEAD** is a pointer (but not a branch) to current active branch



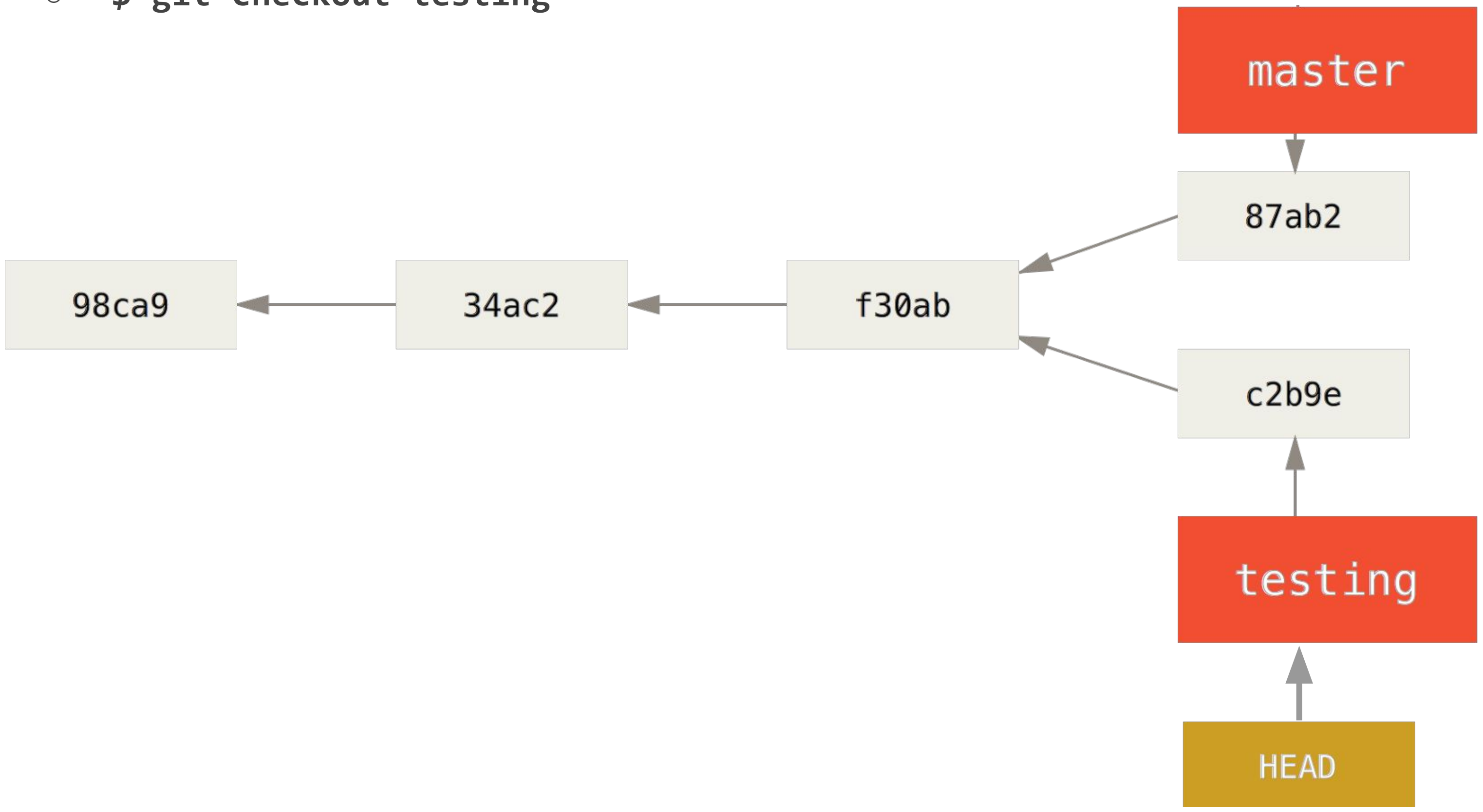
git branch — what is

- **is like a new pointer to the current commit**
  - **master is a branch that is tipically created by default**
  - **HEAD** is a pointer (but not a branch) to current active branch
  - it allows to diverge paths of development



git branch — switching

- it is possible to switch between branches
  - HEAD pointer is changed
  - `$ git checkout testing`

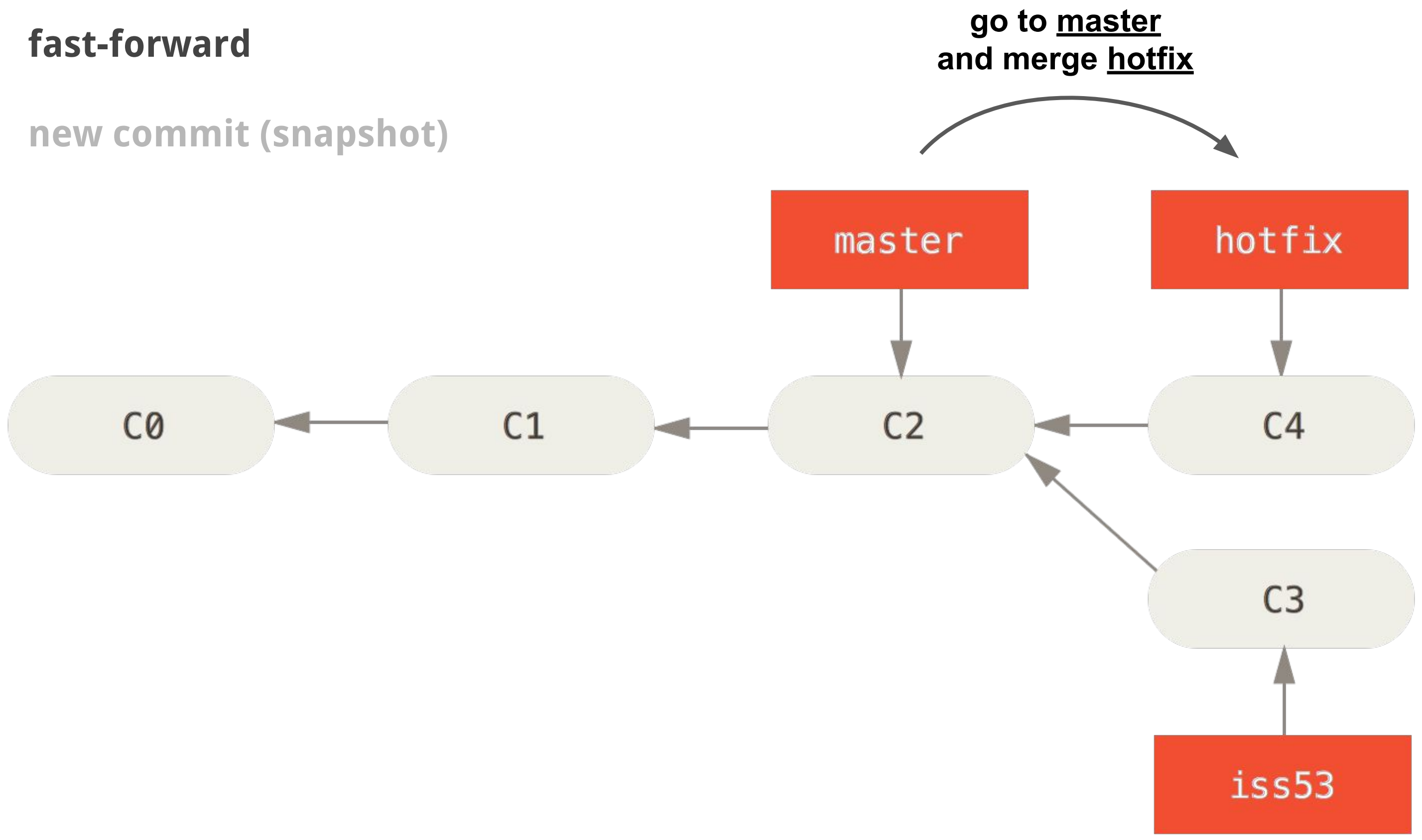


git branch — merging

- two ways of merging
  - fast-forward
  - new commit (snapshot)

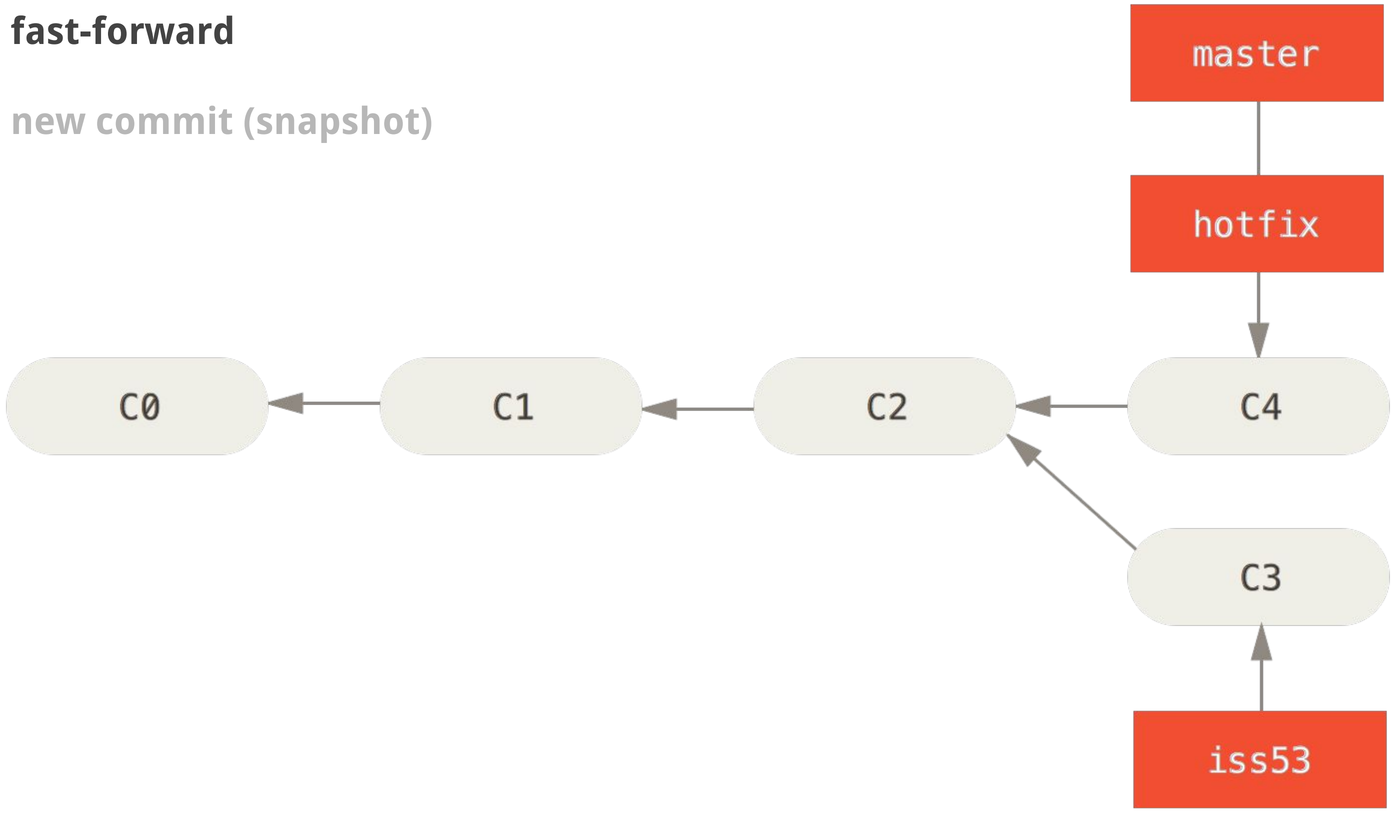
git branch — merging

- two ways of merging
  - fast-forward
  - new commit (snapshot)



git branch — merging

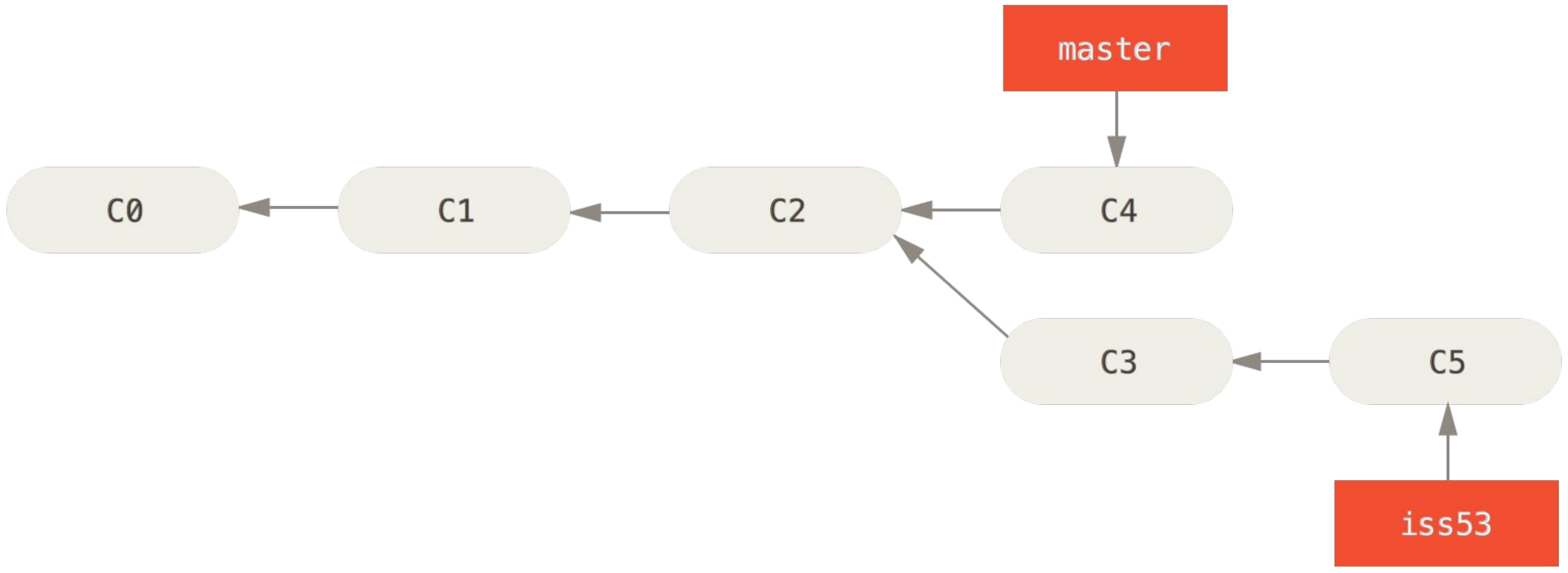
- two ways of merging
  - **fast-forward**
  - new commit (snapshot)



git branch — merging

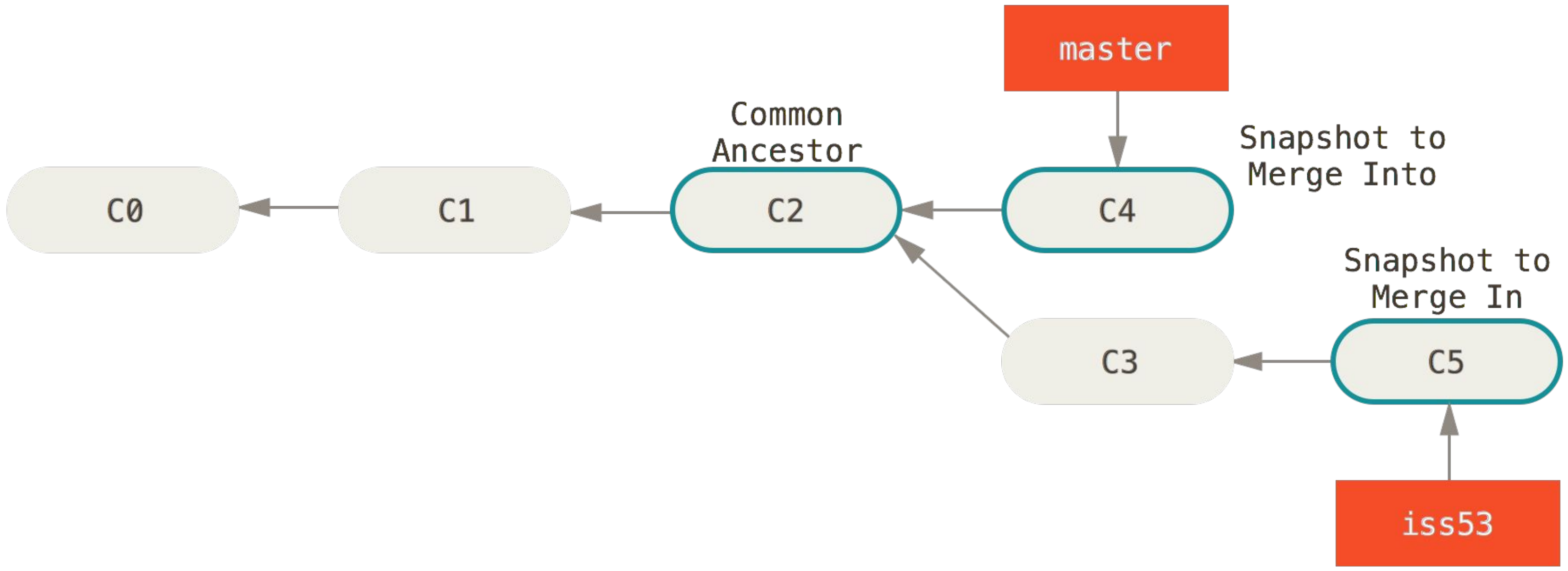
- two ways of merging
  - fast-forward
  - new commit (snapshot)

go to master
and merge iss53



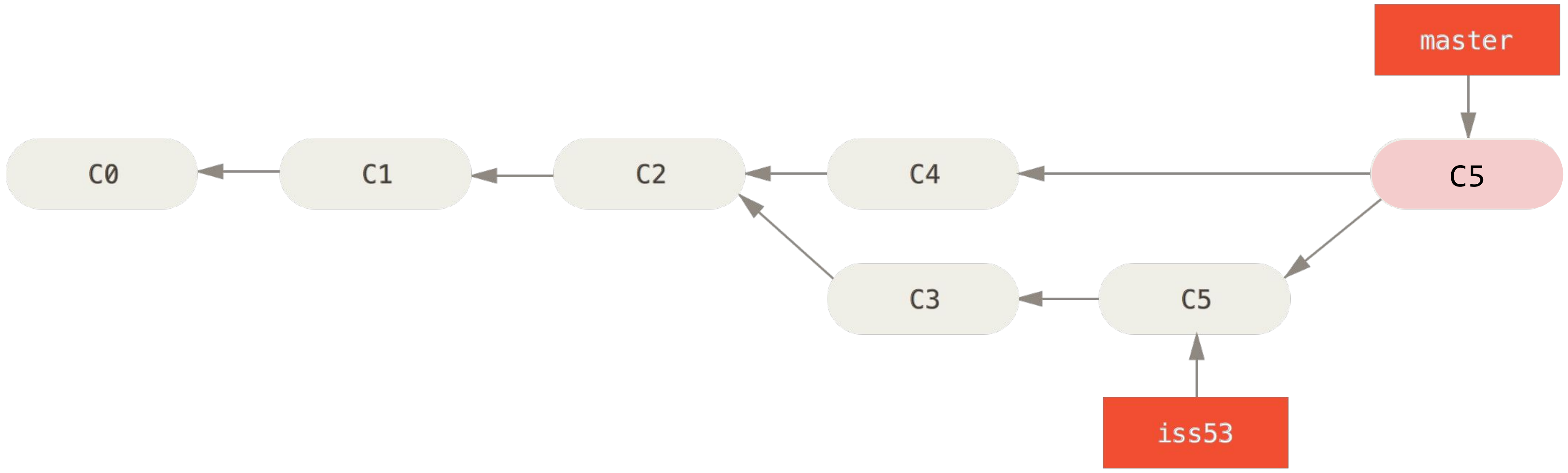
git branch — merging

- two ways of merging
  - fast-forward
  - new commit (snapshot)



git branch — merging

- two ways of merging
  - fast-forward
  - new commit (snapshot)



git branch — merging conflicts

- merges can have conflicts
    - they appear when the same part of files is changed in both branches

index.html master

```
<div>
     content
</div>
```

index.html testing

```
<div>
     new content
</div>
```

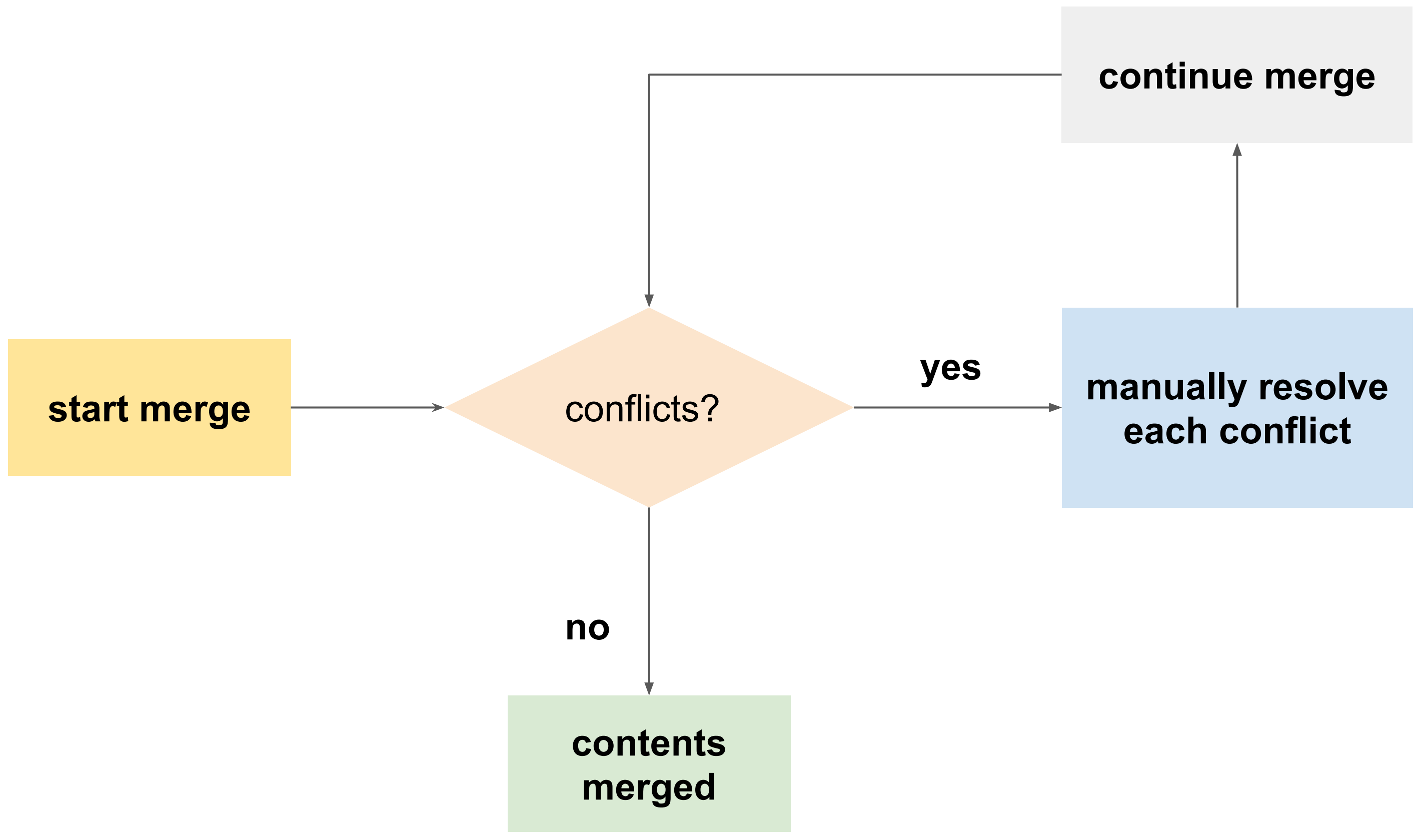
git branch — merging conflicts

- **git annotates conflicts**
    - **requires human intervention to resolve them**
    - **human chooses which one he wants and saves the file**
    - **human commits changes and tells git to continue**

index.html merge testing into master

```
<div>
<<<<<<< HEAD:index.html
     content
=============
     new content
>>>>>>> testing:index.html
</div>
```

git branch — merge flow



git branching

in practice

Hackathonners

git — branch

- create a new branch from current commit
  - `$ git checkout -b <name>`

- delete a branch
  - `$ git branch -D <name>`

- switch branches
  - `$ git checkout <name>`

git — merge

- merge a given branch into current one
    - `$ git merge <branch>`

git remote

in practice

Hackathonners

git — remotes

- manage remote repositories (remote servers)
    - `$ git remote [-v]`
        - `lists configured remote servers`
        - `origin is the most common name of primary server`
    - `$ git remote add <name> <url>`
        - `adds a remote server to current local repository`

git — remote

- push changes to remote server
    - `$ git push <remote> <branch>`
- fetch and merge changes from remote server to current version
    - `$ git pull <remote> <branch>`
- fetch changes
    - `$ git fetch <remote> <branch>`

git — more info

- **online book & try it**
  - http://git-scm.com/book
  - https://try.github.io/levels/1/challenges/1
- **cheat-sheet**
  - https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
- **git remote servers**
  - https://github.com (public repositories)
  - https://gitlab.com (private repositories and teams)
  - https://bitbucket.com (private repositories)

questions?

Hackathonners

Thank you

Francisco Neves
contact@francisconeves.com

GitHub, Twitter: **@fntneves**

Hackathonners