TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

SWEN 225 : Software Design

Collaborative Software Development with GIT

Jens Dietrich **(jens.dietrich@vuw.ac.nz)**

Overview

- motivation
- history
- introducing git
- basic workflows
- branching and merging
- tagging
- misc: rewriting history, forking and pull requests

Trends: Exploding Size of Code Bases

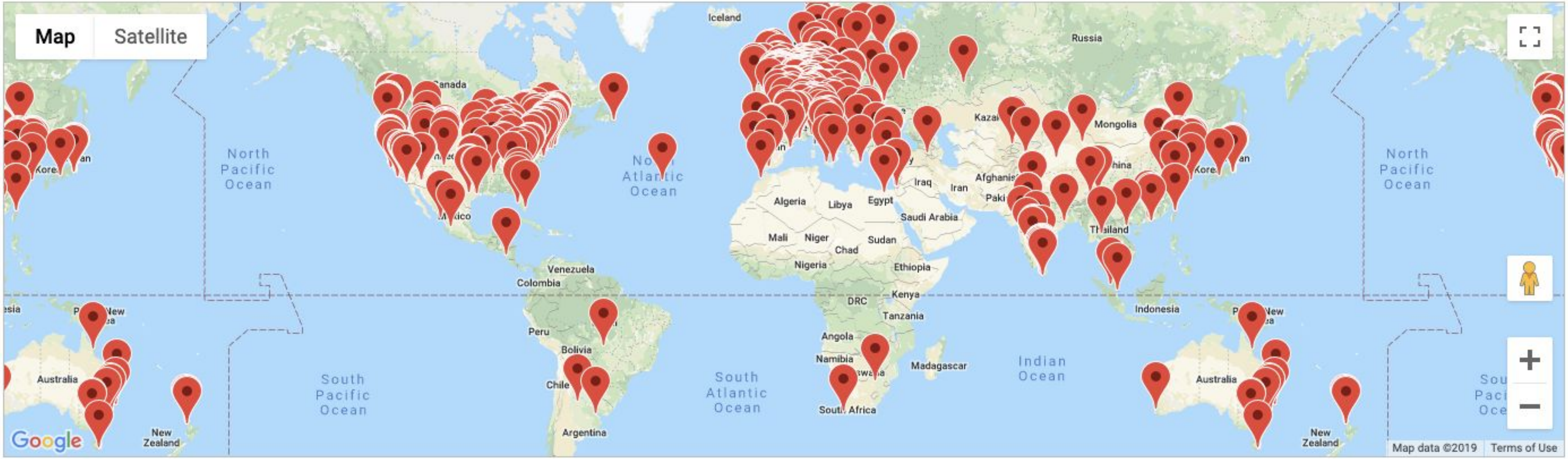
- Boeing 787: 6.5 MLOC behind its avionics and online support systems
- Google Chrome: 6.7 MLOC
- Android: 12-15 MLOC
- Large Hadron Collider: 50 MLOC
- Google services: 2 GLOC

source (2017): https://www.visualcapitalist.com/millions-lines-of-code

Trend: Large Team Sizes

- 2008 Windows 7 -- ca 1,000 developers in 23 groups (https://www.theguardian.com/technology/blog/2008/aug/19/howmanypeoplemakewindows7)
- Kubernetes -- 2,239 contributors (https://github.com/kubernetes/kubernetes , 9 August 19)
- Rust -- 2,442 contributors (https://github.com/rust-lang/rust, 9 August 19)
- NodeJS -- 2,513 contributors (https://github.com/nodejs/node, 9 August 19)
- TensorFlow -- 2,122 contributors (https://github.com/tensorflow/tensorflow, 9 August 2019)
- https://github.com/tensorflow/tensorflow/graphs/contributors

Trend: Distributed Teams



Apache Committers Map https://community.zones.apache.org/map.html (9 August 2019)

Use Cases: what we need and want

- coordinate and integrate work of members of large distributed teams
- a time machine: go back to any state (in case things got messed up)
- audit changes (who, what and when)
- separate development on new features, ability to integrate this back into main product, spikes
- nonfunctional: scalable, secure, fault-tolerant

Version Control Systems

- started to be developed in the 70 ties
- commercial and open-source products became widely used in the 90ties, examples: CVS and Subversion
- Git started by Linus Torvalds in April 2005
- widely used, pushed by GitHub
- similar to Git: Mercurial (hg) started in 2005, and Git, also used in large projects (facebook, openjdk)
- bitbucket phased out support for hg in 2020

Dimensions

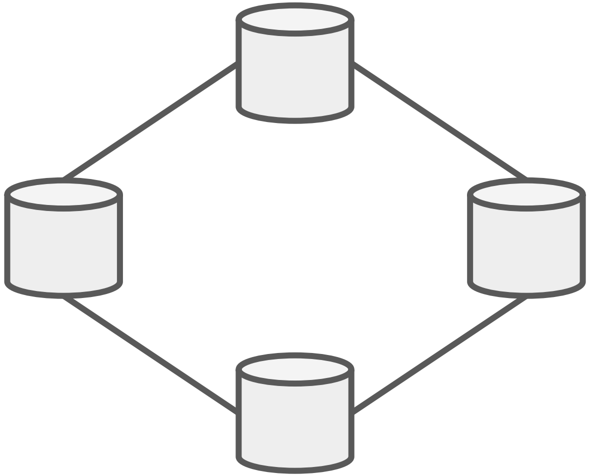
- topology: central client-server vs distributed flexible (user-defined) topology
- concurrency control: prevent conflicts (***"lock"***) vs deal with conflicts ("***merge"***)
- **Git is distributed, and uses merge**
- this scales better, and is better aligned with modern development workflows

Hosting Services

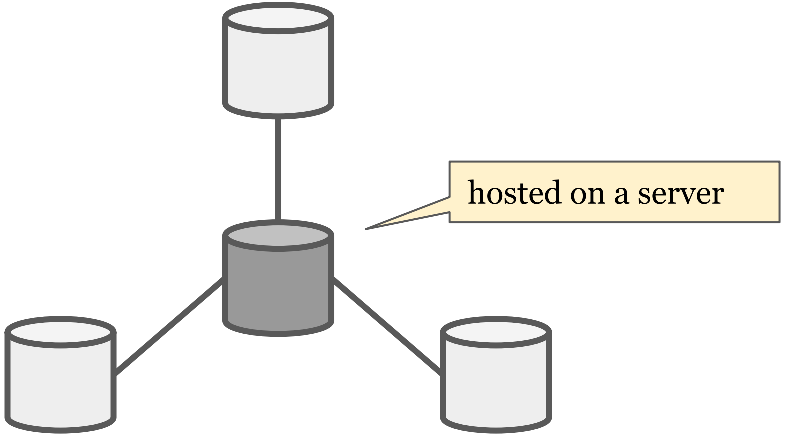
- while Git is distributed (and not client-server), it is often used in a client-server like configuration
- here, a central repository is used (hosted on a server), and local repositories (on clients) connect to it
- there are services that provide Git repo hosting, including GitHub and bitbucket, and VUWs internal GitLab system

note: the database managing code and related resources is called a **repository** (repo for short)

Distributed Repositories



possible

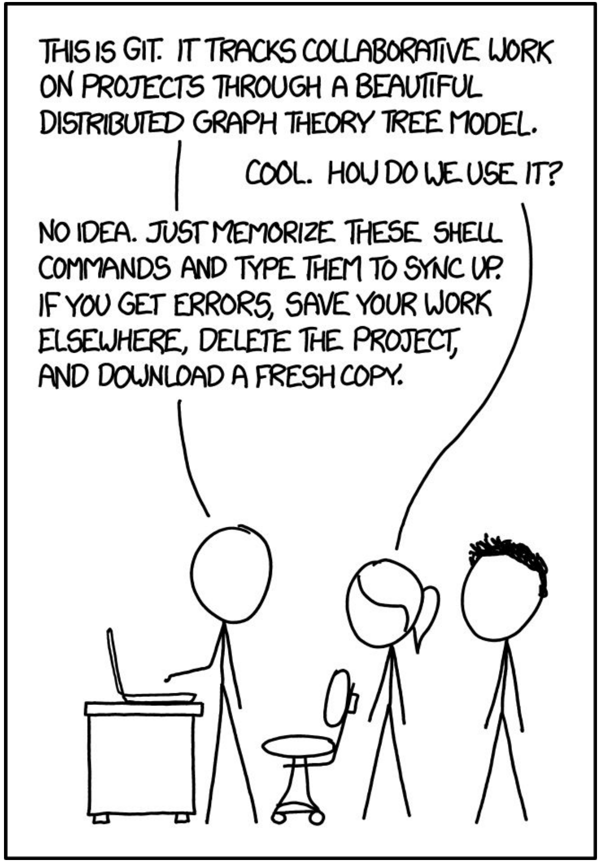


hosted on a server

common

GIT Clients

- CLI - run `git` commands from the terminal (used in this lecture)
- dedicated Git clients (selection):
  - sourcetree -- by Atlassian (bitbucket, etc), also supports hg, free
  - GitHub desktop -- by the provider of GitHub, free
  - Git Kraken -- independent commercial
  - .. many more
- IDE integrated
  - most IDEs (Eclipse, IntelliJ, VisualStudio) support GIT, either native or via plugins


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

https://xkcd.com/1597/

Git Structure

`working tree`
`(aka working directory)`

the local file system, basically the content of the project folder under version control except **`.git`**

`staging area`
`(aka index)`

tracked files: ready to be added to the repository

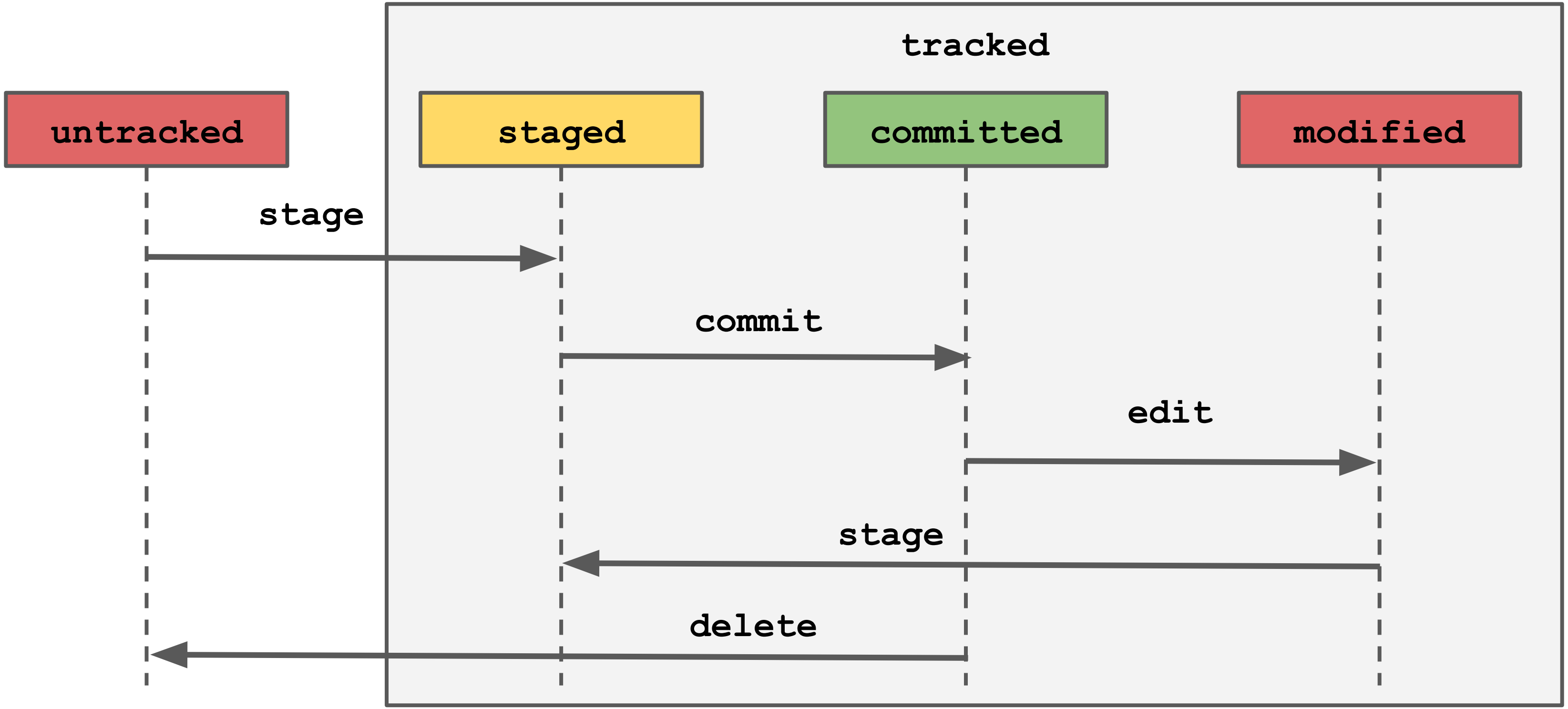
`local repository`

consists of commits -- development snapshots, located in the (hidden) **`.git`** folder

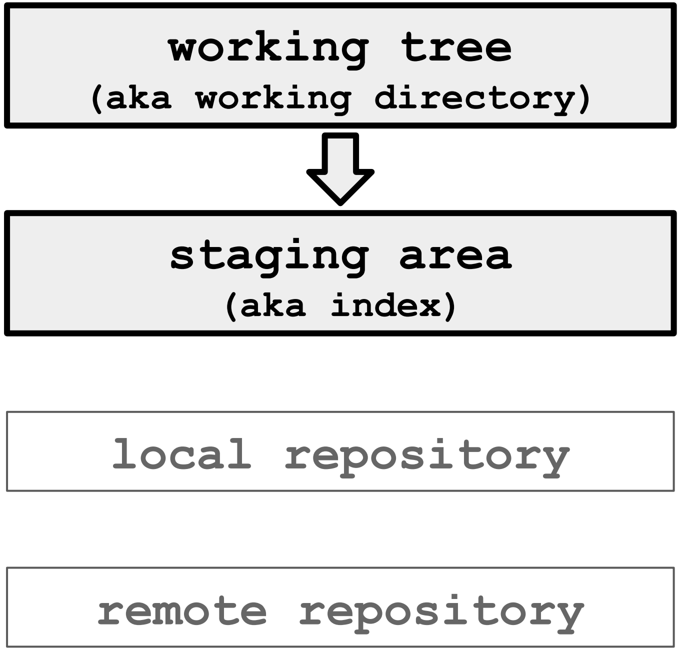
`remote repository`

remote version of repository, can be synchronised with local repository, identified by a URL like **`git@gitlab.ecs.vuw.ac.nz:jens/git101.git`** or **`https://gitlab.ecs.vuw.ac.nz/jens/git101.git`**

File States


tracked
untracked
staged
committed
modified
stage
commit
edit
stage
delete

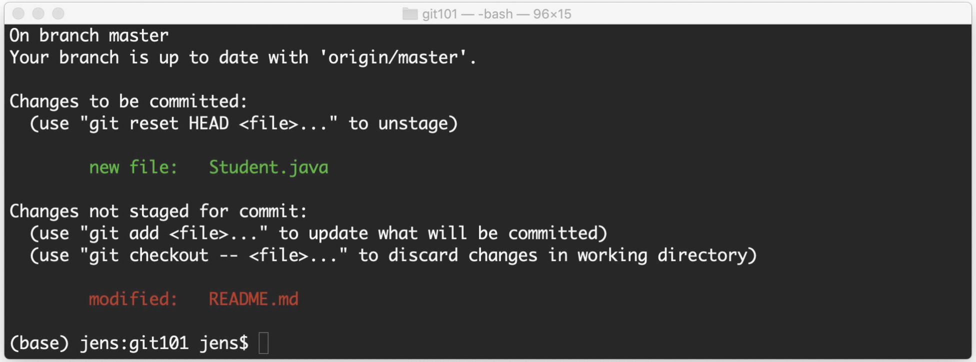
Staging



`git status` -- display state of working tree and staging area

`git add <file>` -- stage file

`git add .` -- stage everything



```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Student.java

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:   README.md

(base) jens:git101 jens$ 
```

Ignoring Files

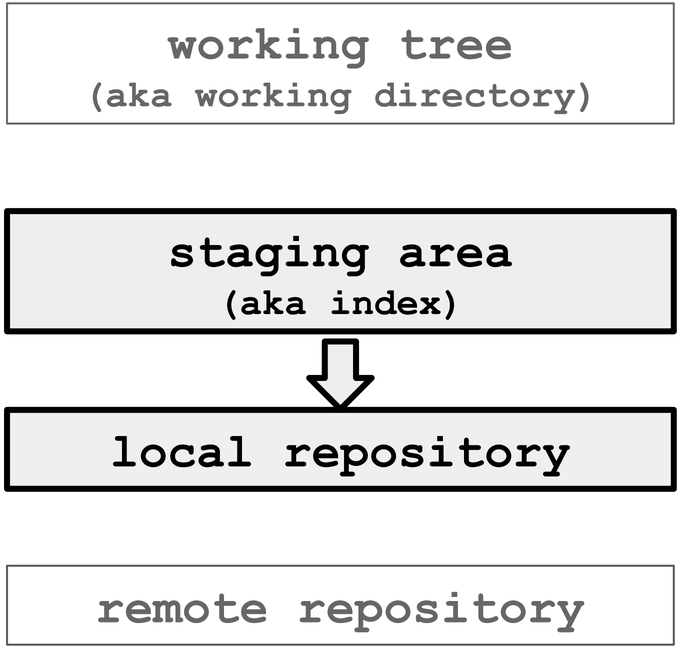
- not every file should be under version control !
- compiled code (example: `*.class`) should not be in the repo -- they can be re-created easily from shared sources, and keeping source and compiled code increases the chances of them being inconsistent
- build tool target folders (ant `build/`, maven `target/`) should be excluded
- IDE metadata (example: Eclipse `.project` and `.classpath`), sharing them means to enforce IDE settings on other users
- sharing IDE data can sometimes make sense if all team members use the same IDE and project settings (e.g. `.classpath` contains the build path for Eclipse projects)
- repository size consideration: many providers have size quotas, and therefore large data file should not be in the repo

Ignoring Files with Git

- create a **`.gitignore`** file in project folder with a list of files to be excluded from staging
- global version: **`~/.gitignore_global`** -- applies to all projects
- can exclude entire folders and use wildcards (**`*`**)
- https://www.gitignore.io/ -- service to create **`.gitignore`** files
- example (recommended exclusions for Java / Maven projects):

  https://www.gitignore.io/api/osx,maven,eclipse

Committing



working tree

(aka working directory)

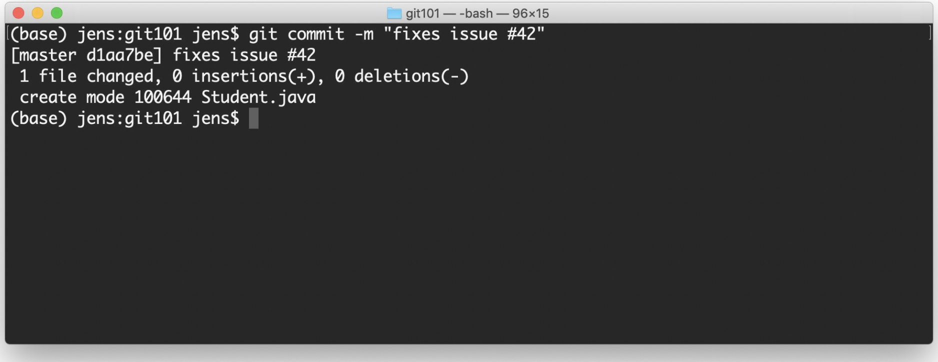
staging area

(aka index)

local repository

remote repository

```
git commit -m “<commit message>”
```

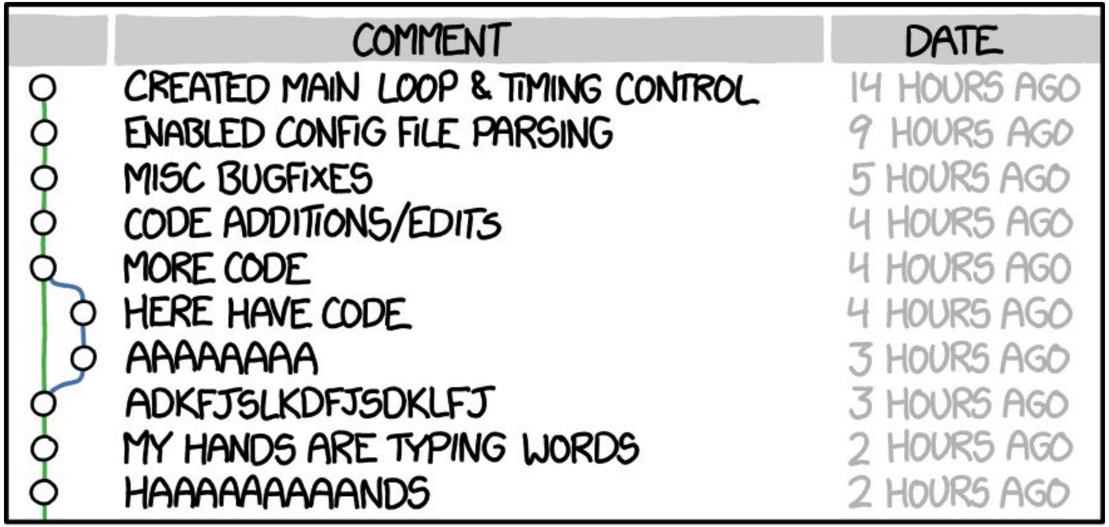


```
(base) jens:git101 jens$ git commit -m "fixes issue #42"
[master d1aa7be] fixes issue #42
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Student.java
(base) jens:git101 jens$ 
```

Commit Messages

- descriptive summaries of change
- many repository hosting services integrate repositories with issue tracking systems, scan messages and perform actions, such as closing or updating issues (bug reports, new feature requests)
- commits without messages should be avoided

Commit Messages



COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAANDS	2 HOURS AGO

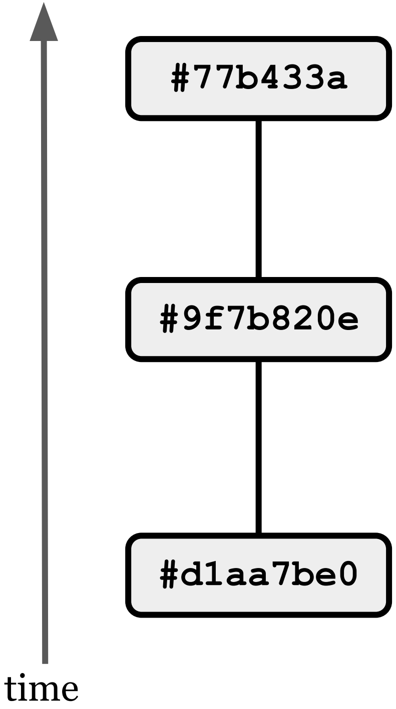
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

https://xkcd.com/1296/

The Structure of the Repository

- a repository consists of **linked commits** -- it is basically a directed, acyclic graph
- commits are snapshots of the file system
- for efficiency reasons, unchanged files are not saved again and again, but represented using a reference to an older commit
- commits are identified by a computed identifier (using SHA-1)
- all files in a snapshot are checksummed with SHA-1, so Git can track changes

Commits



- commits have pointer to parent commit -- the previous snapshot
- many GUI clients only visualise this with an edge: the child is above the parent
- e.g., **`#9f7b820e`** is parent of **`#77b433a`**
- commits have additional metadata:
  - author
  - commit message
  - timestamp

note: SHA-1 hashes are simplified

Connecting to a Remote Repository

- to use a repo for sharing, (one or many) central repos are used
- they are synchronized with the local repo(s)
- they are referred to as remotes
- git command to show remotes: **`git remote -v`**
- once remotes are configured, data can be pushed into the remote, or pulled (fetched) from the remote

Connecting to a Remote Repository

- approach 1: create a remote repository first (with gitlab or similar , then clone it: **`git clone <repo-url>`**

  this will create a local repository connected to a remote, the remote is called the *origin*

- approach 2: create local repository first (**`git init`**), then link it to remote repo:

  ```
  git remote add origin <repo-url>
  ```

Identifying a Remote Repository

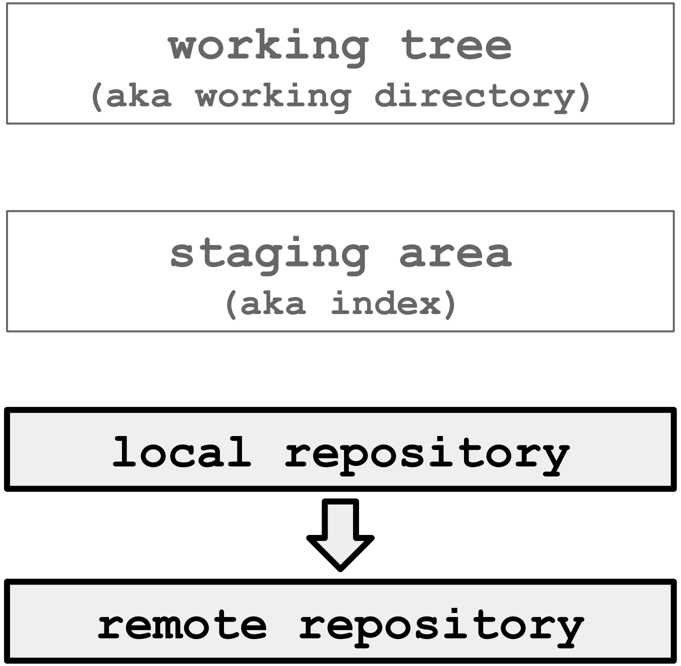
- GitLab / GitHub / Bitbucket provide URLs for cloning on project web pages
- two types of URL:

`git@gitlab.ecs.vuw.ac.nz:jens/git101.git` -- uses SSH, requires keys to be generated, and server needs to have public key, avoid password-based authentication

hint: in gitlab, add keys to Profiles > Settings > SSH keys

`https://gitlab.ecs.vuw.ac.nz/jens/git101.git` -- used HTTPS, requires username / password

Pushing



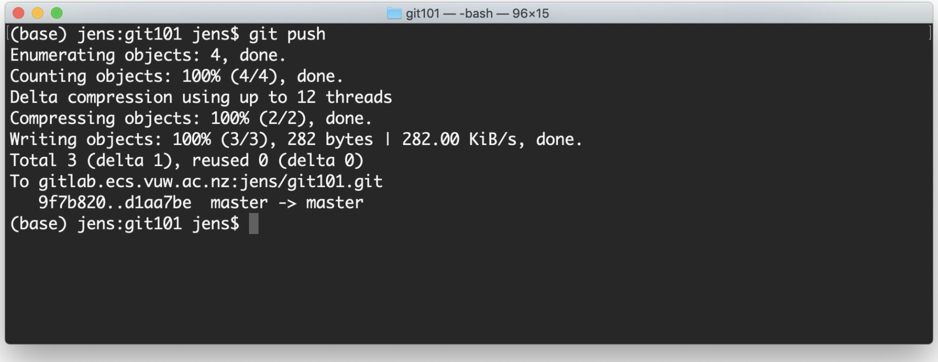
working tree
(aka working directory)

staging area
(aka index)

local repository

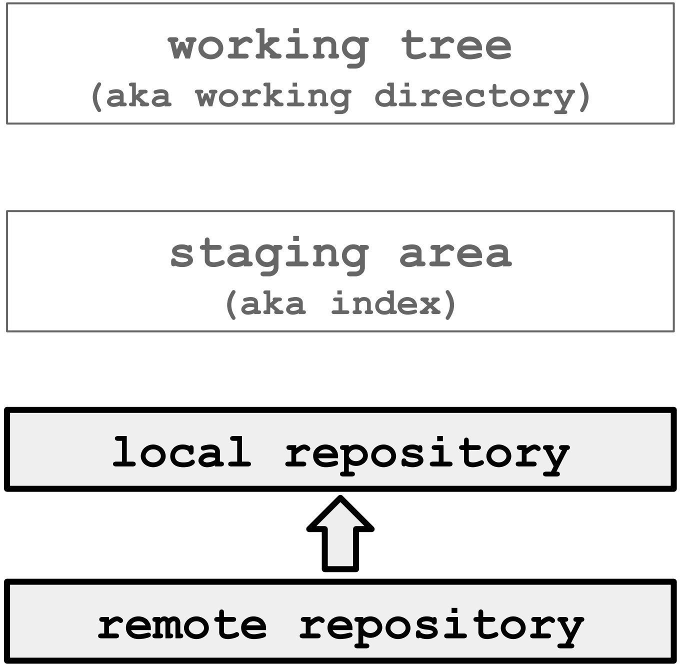
remote repository

`git push`



```
(base) jens:git101 jens$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes | 282.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To gitlab.ecs.vuw.ac.nz:jens/git101.git
   9f7b820..d1aa7be  master -> master
(base) jens:git101 jens$ 
```

Pulling



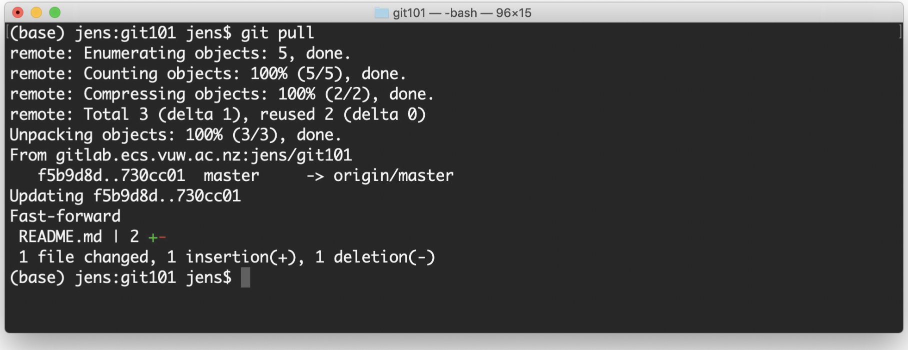
working tree
(aka working directory)

staging area
(aka index)

local repository

remote repository

`git pull`

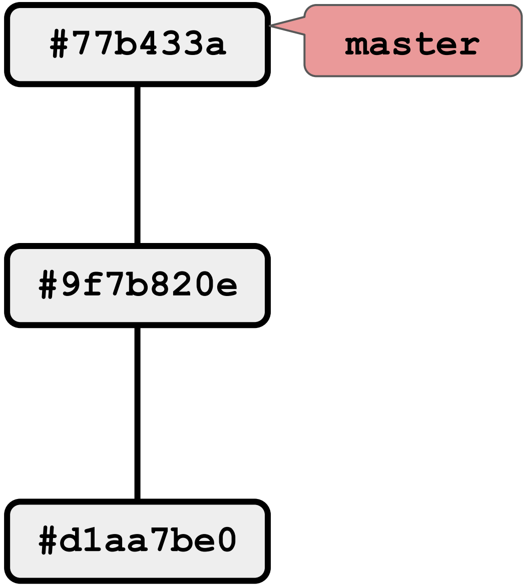


```
(base) jens:git101 jens$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 2 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.ecs.vuw.ac.nz:jens/git101
   f5b9d8d..730cc01  master     -> origin/master
Updating f5b9d8d..730cc01
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) jens:git101 jens$
```

Branching -- Use Cases

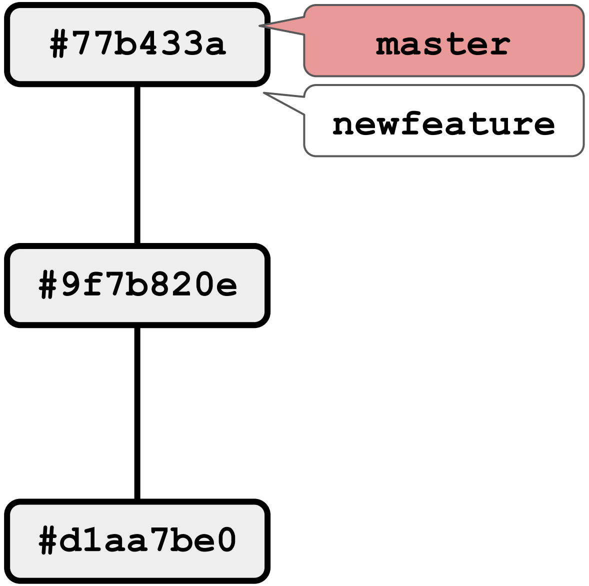
- parallel development in teams, and on different features
- company has a successful product and has released a version
- it starts developing experimental features for the next version, while continuing to develop (for maintenance) the current product version
- branching allows to isolate development of features
- merging is the process of consolidating branches to bring code together

Branching



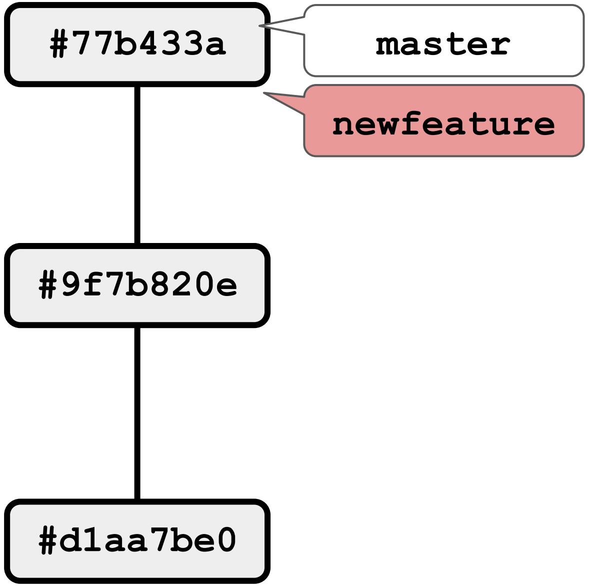
- a branch is a **movable pointer** (label) associated with a commit
- the active branch is marked as the HEAD (GIT clients will highlight it somehow)
- after a new commit, the branch moves to the new commit
- most repos have a master branch -- `git init` creates this

Adding a New Branch



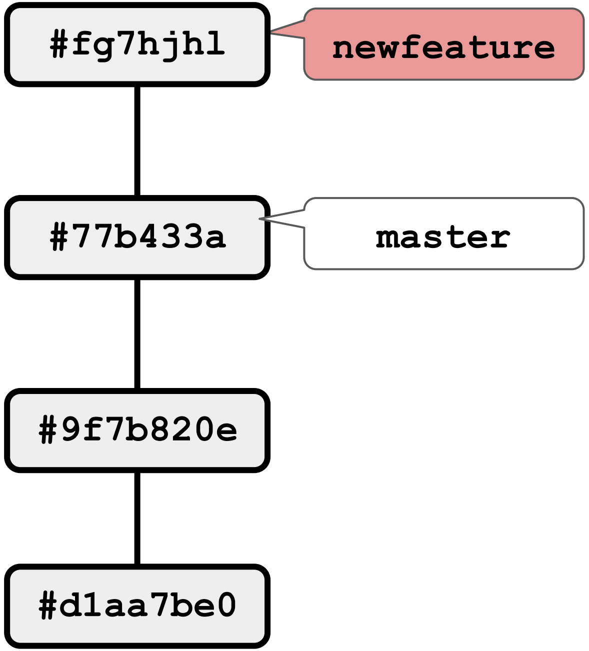
- **`git branch newfeature`**
- new branch created, but **`master`** is still the **`HEAD`**

Switch to Branch



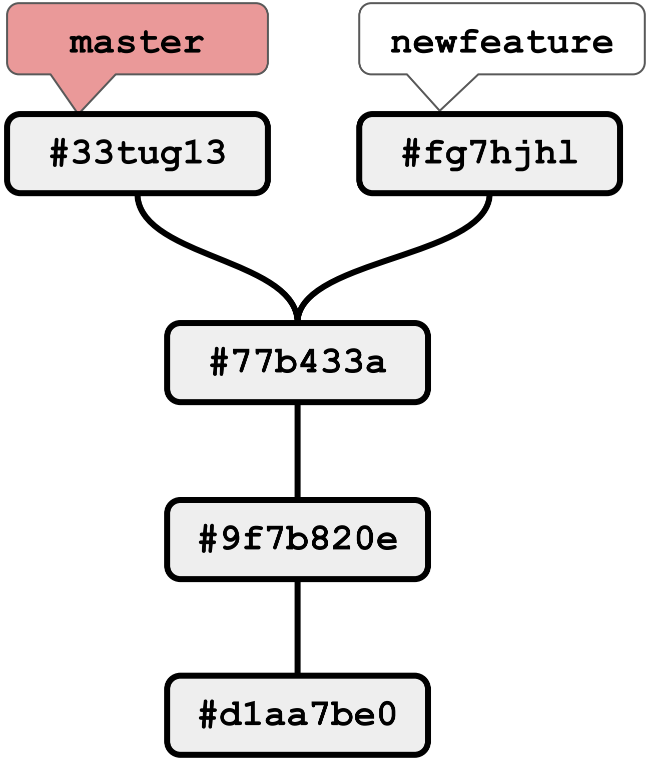
- **`git checkout newfeature`**
- now **`newfeature`** is the **`HEAD`**

Making some Changes, and Commit



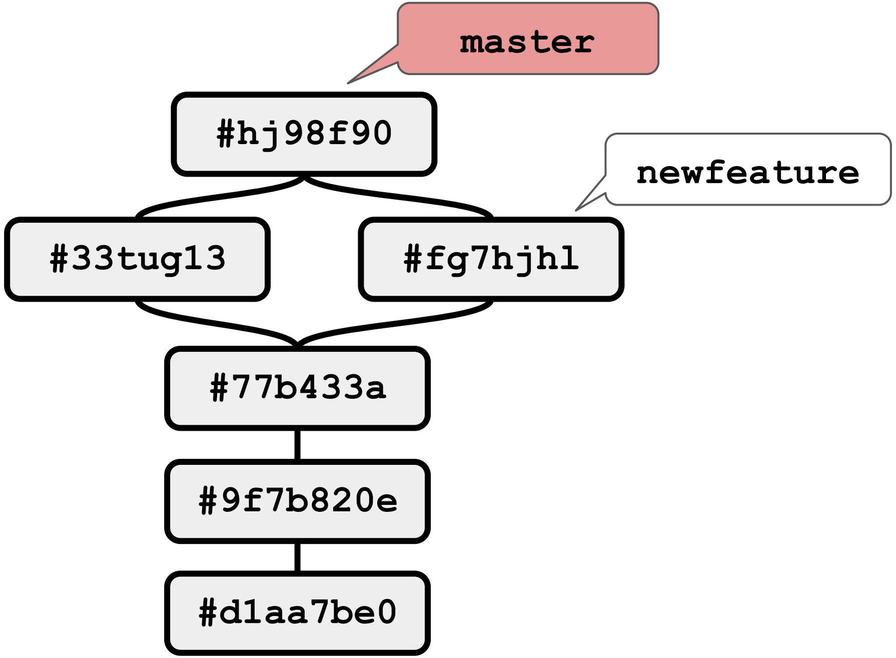
- make some changes
- **`git commit -m '..'`**
- the active branch pointer moves to the new commit

Switch Back to **`master`**, make Changes & Commit



- **`git checkout master`**
- make some changes
- **`git commit -m ‘..’`**

Merging



- **`git checkout master`**
- **`git merge newfeature`**
- a new *merge commit* is automatically created
- the active branch pointer moves to the new commit

Merging

- easy case: in both branches, different files have been modified
- more tricky: conflicting changes to same (non-binary) file
- conflicts must be resolved using special editors
- git highlights conflicts using text syntax:

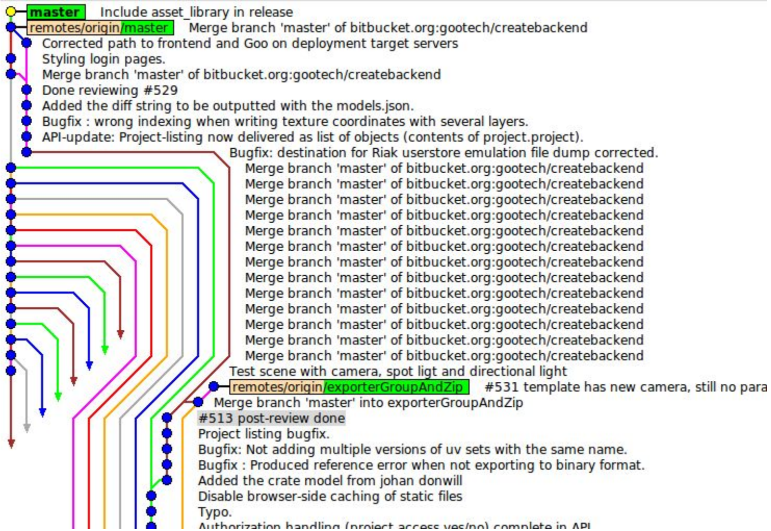
```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>

>>>>>>> iss53:index.html
```

example from:
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

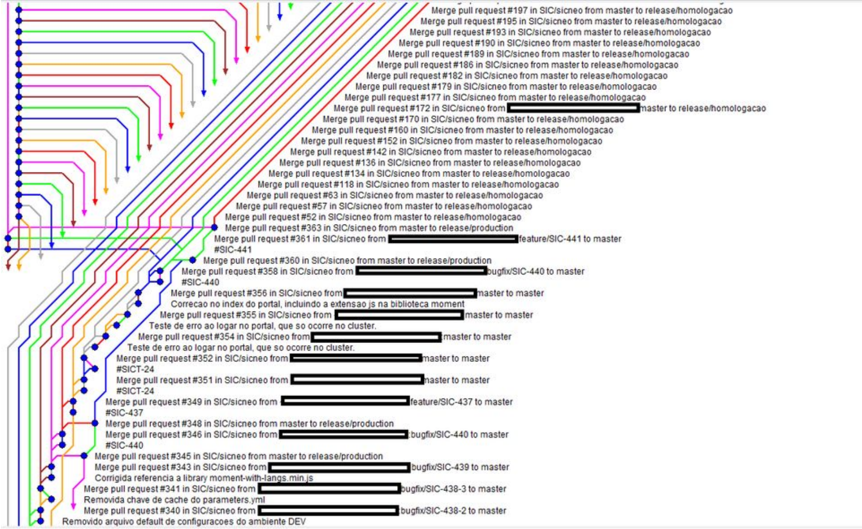
Merging without Branching (Kind-Of)

- even if explicite branching is avoided, there is still a need for merging
- assume two team members (#1 and #2) work on master, and make inconsistent changes to files, then #1 pushes
- this updates the master branch in the remote repo (**`origin/master`**)
- the push by #2 fails as it would override more recent changes
- to resolve this, #2 needs to pull, and merge the pulled branch **`origin/master`** into **`master`**

The Beauty of Merging: Git Rainbows



```
master  Include asset_library in release
remotes/origin/master  Merge branch 'master' of bitbucket.org:gootech/createbackend
Corrected path to frontend and Goo on deployment target servers
Styling login pages.
Merge branch 'master' of bitbucket.org:gootech/createbackend
Done reviewing #529
Added the diff string to be outputted with the models.json.
Bugfix : wrong indexing when writing texture coordinates with several layers.
API-update: Project-listing now delivered as list of objects (contents of project.project).
Bugfix: destination for Riak userstore emulation file dump corrected.
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Merge branch 'master' of bitbucket.org:gootech/createbackend
Test scene with camera, spot ligt and directional light
remotes/origin/exporterGroupAndZip  #531 template has new camera, still no para
Merge branch 'master' into exporterGroupAndZip
#513 post-review done
Project listing bugfix.
Bugfix: Not adding multiple versions of uv sets with the same name.
Bugfix : Produced reference error when not exporting to binary format.
Added the crate model from johan donwill
Disable browser-side caching of static files
Typo.
Authorization handling (project access yes/no) complete in API
```

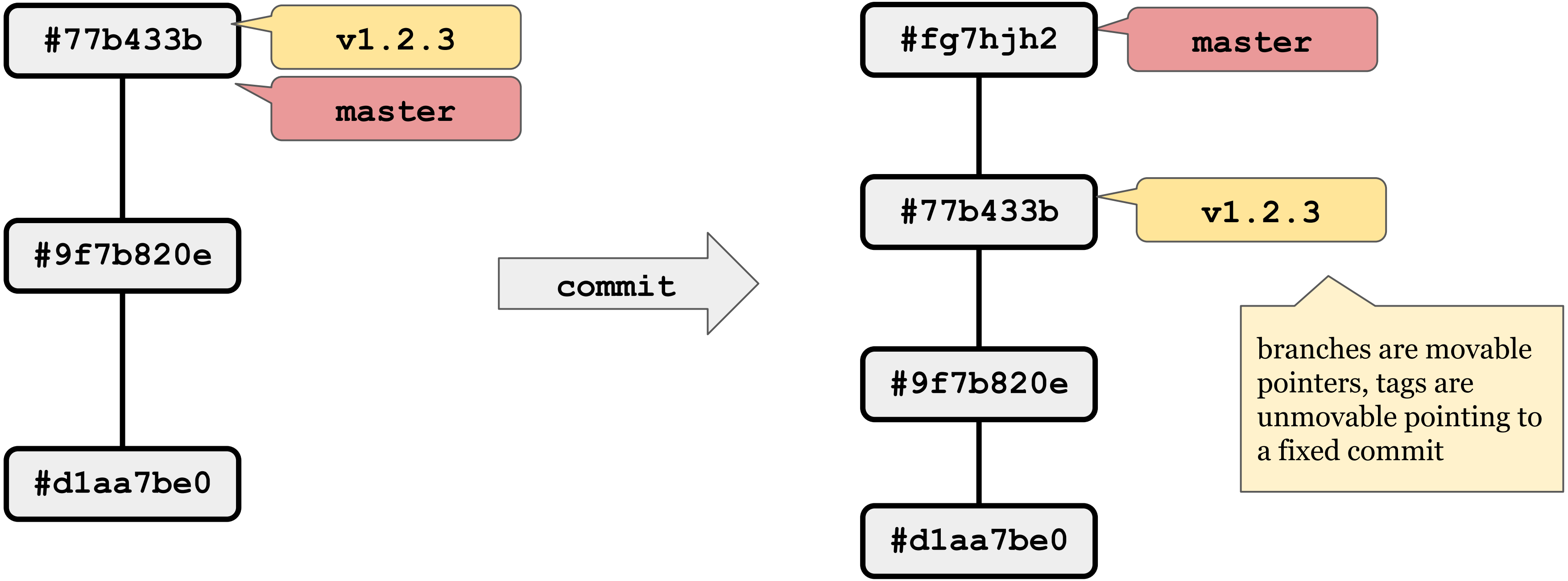


```
Merge pull request #197 in SIC/sicneo from master to release/homologacao
Merge pull request #195 in SIC/sicneo from master to release/homologacao
Merge pull request #193 in SIC/sicneo from master to release/homologacao
Merge pull request #190 in SIC/sicneo from master to release/homologacao
Merge pull request #189 in SIC/sicneo from master to release/homologacao
Merge pull request #186 in SIC/sicneo from master to release/homologacao
Merge pull request #182 in SIC/sicneo from master to release/homologacao
Merge pull request #179 in SIC/sicneo from master to release/homologacao
Merge pull request #177 in SIC/sicneo from master to release/homologacao
Merge pull request #172 in SIC/sicneo from [redacted]master to release/homologacao
Merge pull request #170 in SIC/sicneo from master to release/homologacao
Merge pull request #160 in SIC/sicneo from master to release/homologacao
Merge pull request #152 in SIC/sicneo from master to release/homologacao
Merge pull request #142 in SIC/sicneo from master to release/homologacao
Merge pull request #136 in SIC/sicneo from master to release/homologacao
Merge pull request #134 in SIC/sicneo from master to release/homologacao
Merge pull request #118 in SIC/sicneo from master to release/homologacao
Merge pull request #63 in SIC/sicneo from master to release/homologacao
Merge pull request #57 in SIC/sicneo from master to release/homologacao
Merge pull request #52 in SIC/sicneo from master to release/homologacao
Merge pull request #363 in SIC/sicneo from master to release/production
Merge pull request #361 in SIC/sicneo from [redacted]feature/SIC-441 to master
#SIC-441
Merge pull request #360 in SIC/sicneo from master to release/production
Merge pull request #358 in SIC/sicneo from [redacted]bugfix/SIC-440 to master
#SIC-440
Merge pull request #356 in SIC/sicneo from [redacted]master to master
Correcao no index do portal, incluindo a extensao js na biblioteca moment
Merge pull request #355 in SIC/sicneo from [redacted]master to master
Teste de erro ao logar no portal, que so ocorre no cluster.
Merge pull request #354 in SIC/sicneo from [redacted]master to master
Teste de erro ao logar no portal, que so ocorre no cluster.
Merge pull request #352 in SIC/sicneo from [redacted]master to master
#SICT-24
Merge pull request #351 in SIC/sicneo from [redacted]master to master
#SICT-24
Merge pull request #349 in SIC/sicneo from [redacted]feature/SIC-437 to master
#SIC-437
Merge pull request #348 in SIC/sicneo from master to release/production
Merge pull request #346 in SIC/sicneo from [redacted]bugfix/SIC-440 to master
#SIC-440
Merge pull request #345 in SIC/sicneo from master to release/production
Merge pull request #343 in SIC/sicneo from [redacted]bugfix/SIC-439 to master
Corrigida referencia a library moment-with-langs.min.js
Merge pull request #341 in SIC/sicneo from [redacted]bugfix/SIC-438-3 to master
Removida chave de cache do parameters.yml
Merge pull request #340 in SIC/sicneo from [redacted]bugfix/SIC-438-2 to master
Removido arquivo default de configuracoes do ambiente DEV
```

Tagging

- a tag points to a particular point in history
- this is usually related to releases
- tags can be annotated (with a message, information about the user) or simple
- **`git tag`** -- list tags
- **`git tag -a v1.2.3 -m "released 1.2.3"`** -- create an annotated tag
- **`git checkout tags/v1.2.3`** -- go back to this version (e.g., to start a new branch)

Tags vs Branches



branches are movable pointers, tags are unmovable pointing to a fixed commit

Re-writing History

- **discard** uncommitted changes: **`git reset --hard`**
- **amending** the last commits: **`git commit -amend`**
- **rebasing**: can remove branches by moving them forward in order to create a linear history
- **squashing** -- version of rebase where multiple commits are combined
- this is a version of re-writing history
- **rewriting history in public / shared repos should be avoided**

Forking and Pull Requests

- popular alternative topology and workflow
- instead of using cloning, a repo is **forked**: a copy is created, **disconnected** from the main repo
- to sync the forked repo with the main repoy, a **pull request** is created that needs approval by the maintainer of the original repository
- this means that the party who made changes does not need to have access to the main repo
- use case: “casual” users fixing bugs, and trying to add them to a project

Issue Tracking

- many repos also offer issue tracking
- issues are opened, assigned to a team member, verified and closed: they have state
- issues are not always bugs, this can also be used for planning, where issues are tasks
- **semantic commit messages** are used to directly interact with the issue tracking system, e.g. the sentence “this closed issue #42” in a commit message would in many systems close issue no 42

Resources

https://git-scm.com/

https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud