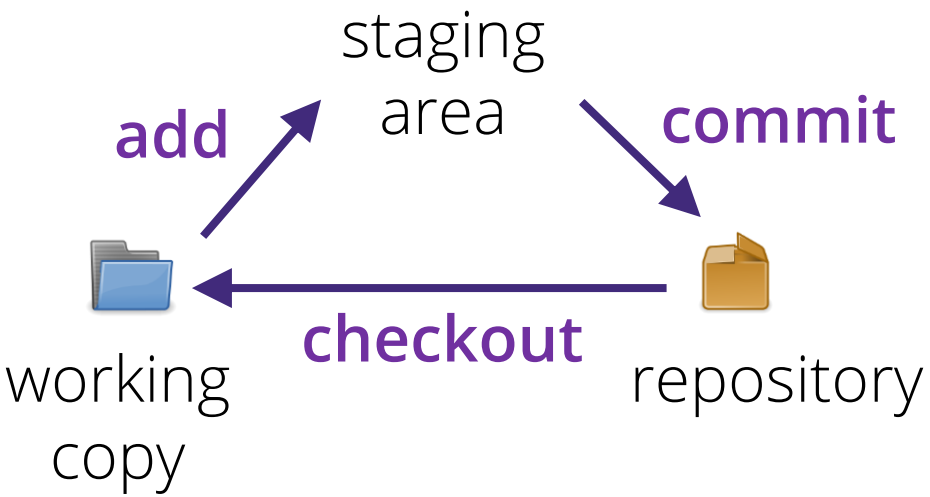
Git 2

COMS10012 Software Tools

remotes

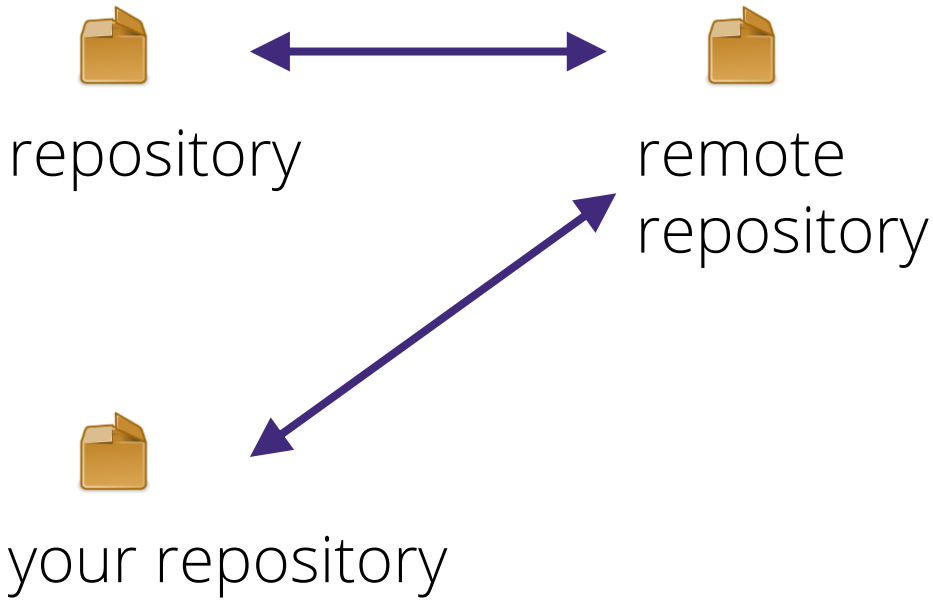
diagram



init **status**

log

multiuser git



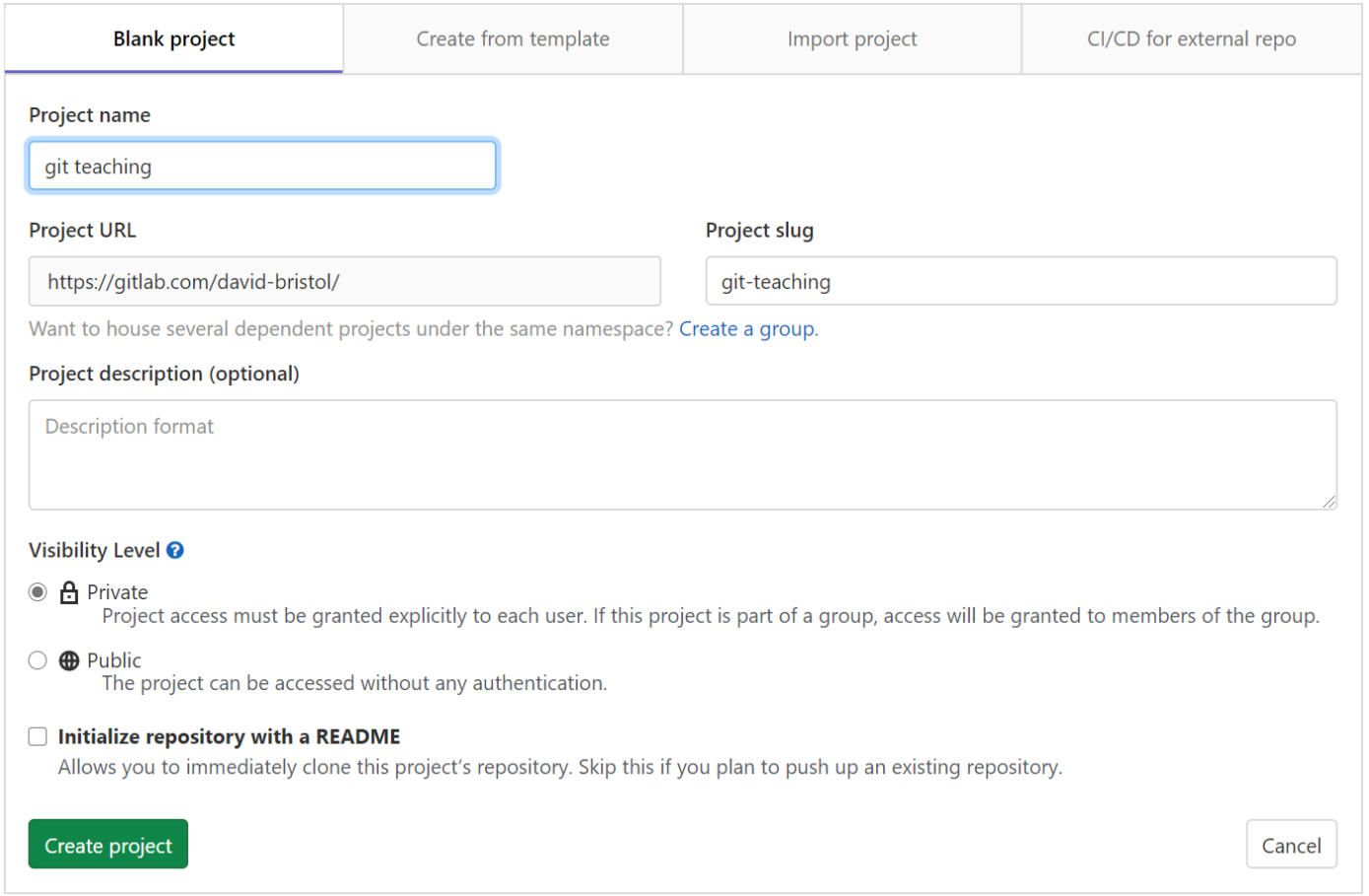
The big 3

GitHub .com

.com

.org

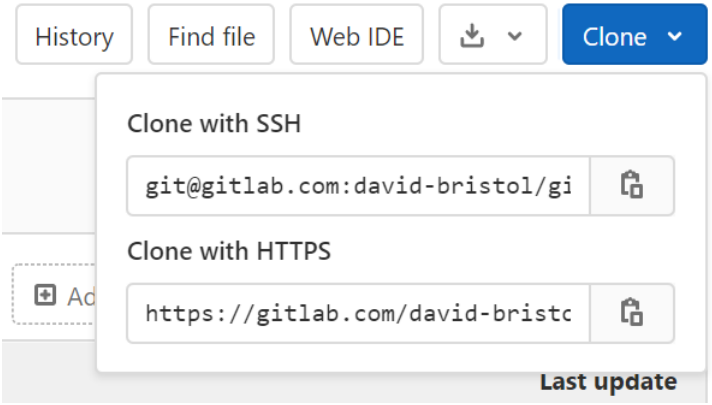
Create new project



Project URLs

`https://gitlab.com/USERNAME/PROJECTSLUG`

`git@gitlab.com:USERNAME/PROJECTSLUG`



Project URLs

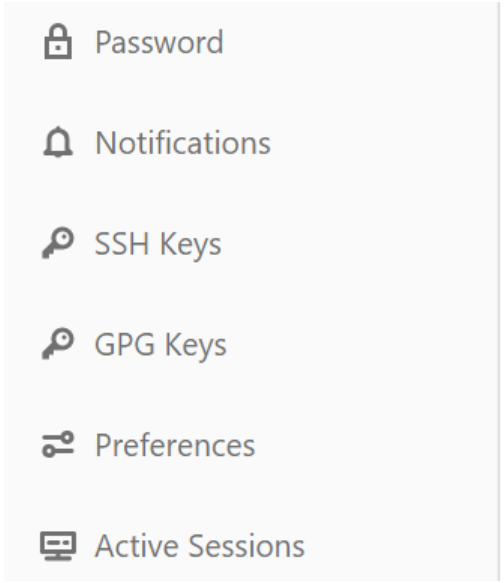
Two ways to access

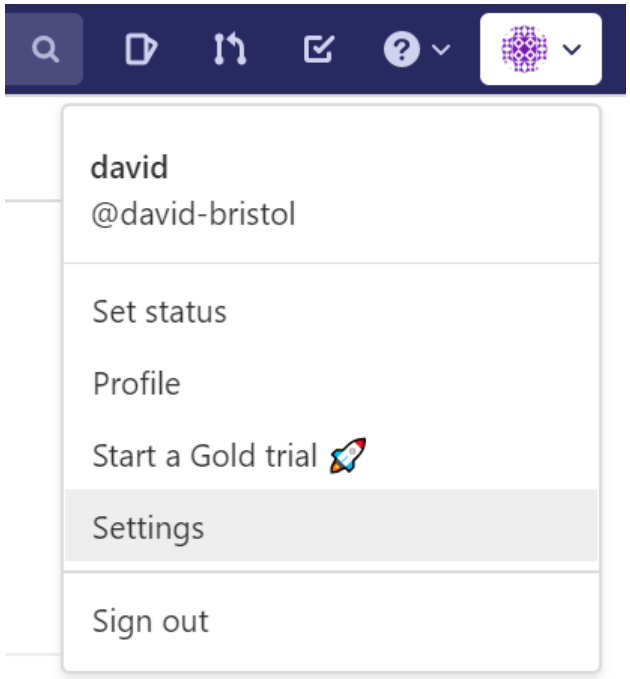
- HTTPS
  for public reads
  if authenticated: username/password
- SSH
  the recommended way for your own projects
  uses public-key crypto

Creating a key

```
$ ssh-keygen -t ed25519
Your identification has been saved in
/home/vagrant/.ssh/id_ed25519.
Your public key has been saved in
/home/vagrant/.ssh/id_ed25519.pub.
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOSSmgCOecNkdCAXbD3K8
eBaAgceHy5ujDiCjwhOIF98
vagrant@alpine310.localdomain
```


Export your public key





Export your public key

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Don't use your private SSH key.

Typically starts with "ssh-ed25519 ..." or "ssh-rsa ..."

Title

e.g. My MacBook key

Give your individual key a title. This will be publically visible.

Expires at

dd/mm/yyyy

Add key

clone the repository

```
$ git clone git@gitlab.com:USERNAME/REPO
$ cd REPO
repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```


remotes

```
$ git remote
origin

$ git remote show origin
* remote origin
  Fetch URL: git@gitlab.com:USER/REPO
  Push  URL: git@gitlab.com:USER/REPO
  HEAD branch: master
  Remote branch:
    master tracked
```

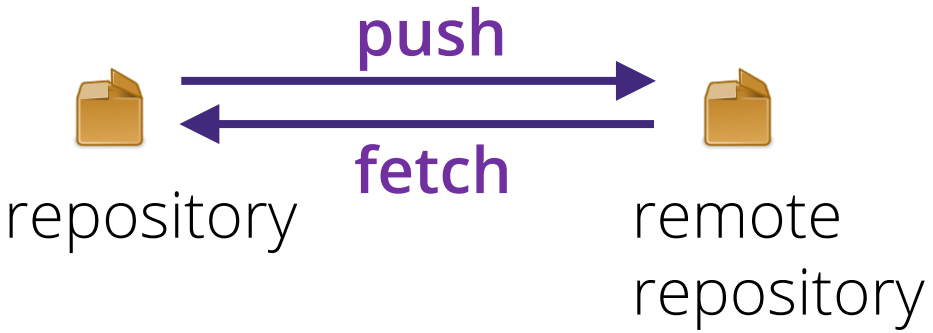

managing remotes

Alternative approach:

1. Create local repo (git init).
2. Create empty repo (no README) on gitlab.
3. `git remote add origin git@gitlab...`

teamwork

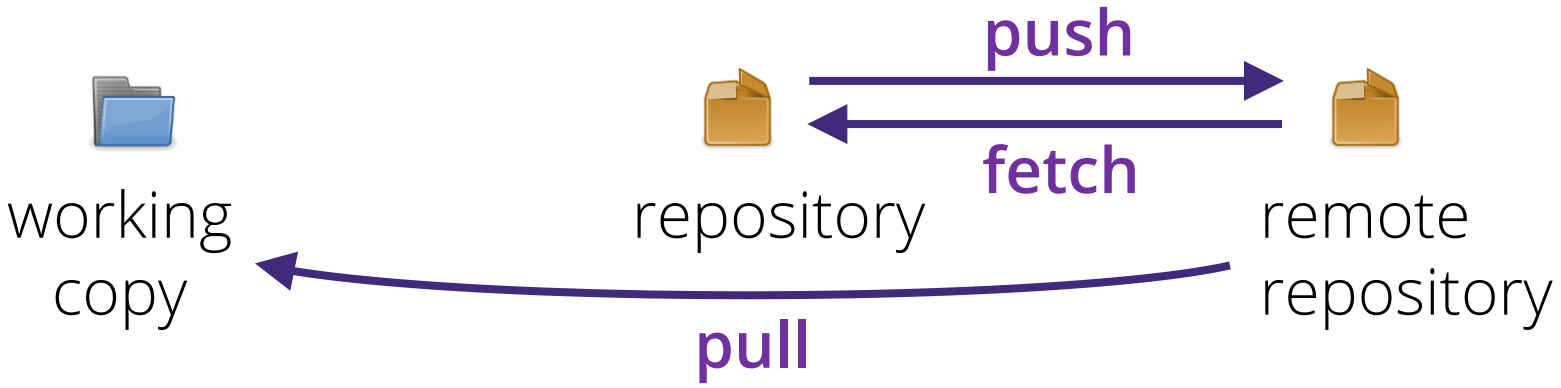
push and fetch



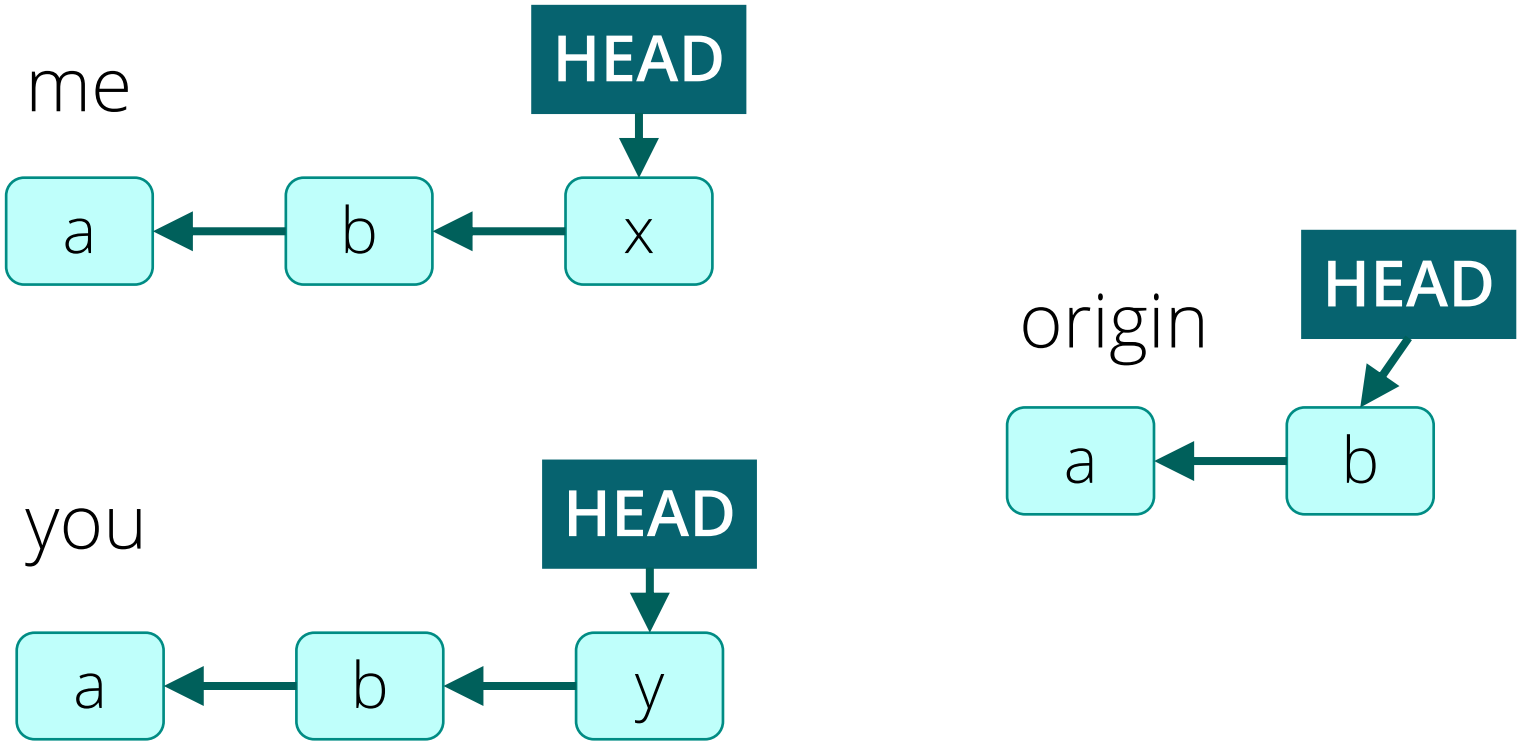
status

```
$ git status
On branch master
Your branch is ahead of 'origin/master'
by 1 commit.
  (use "git push" to publish your local
commits)
```

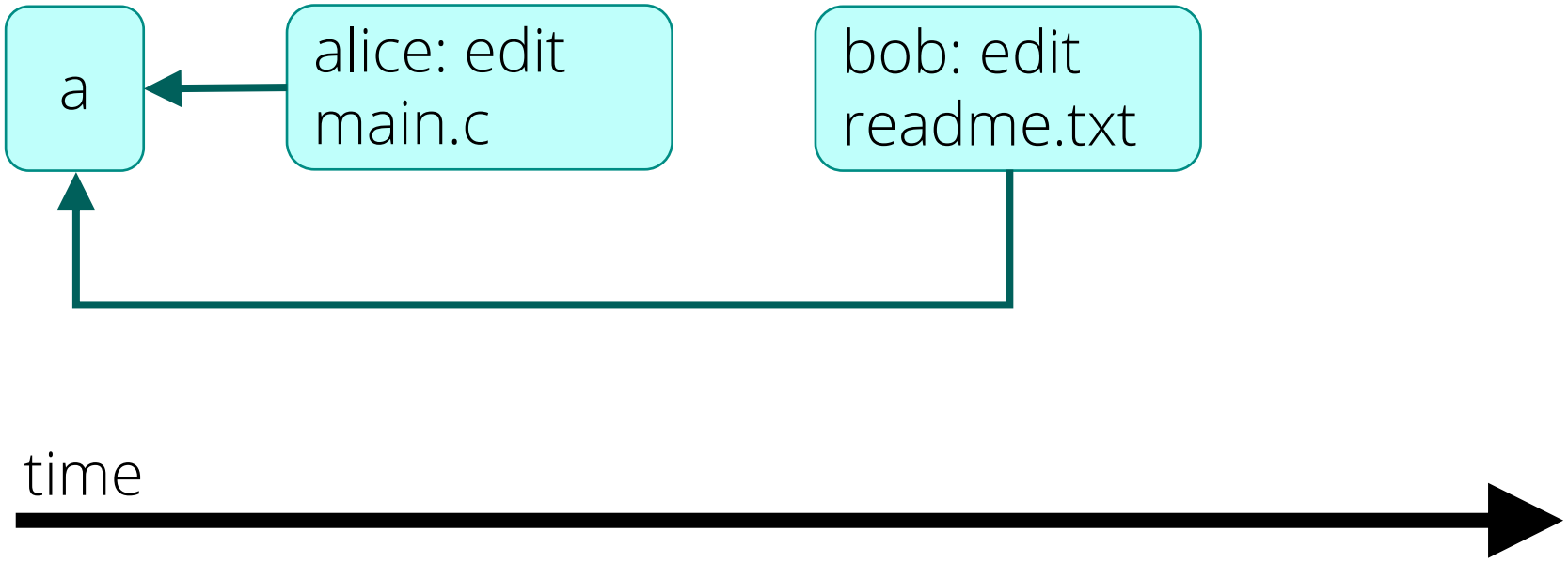

pull


push
fetch
working
copy
repository
remote
repository
pull

fetch is safe

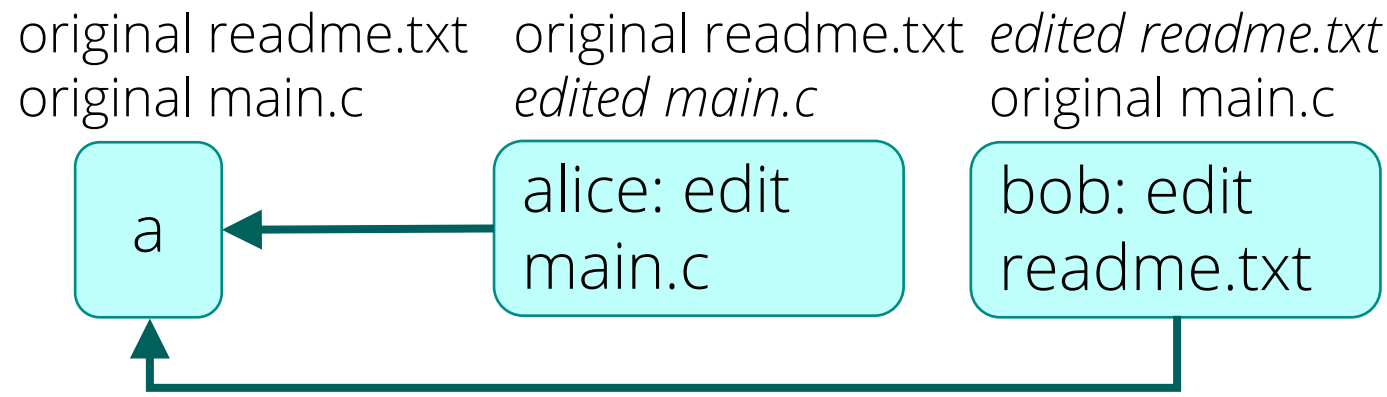


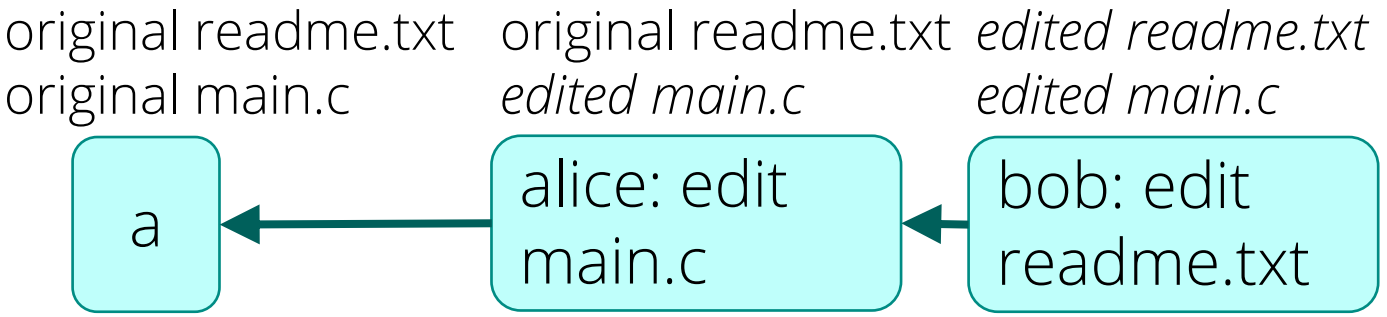
fake conflicts



Different files edited: can fast-forward.

fast-forward



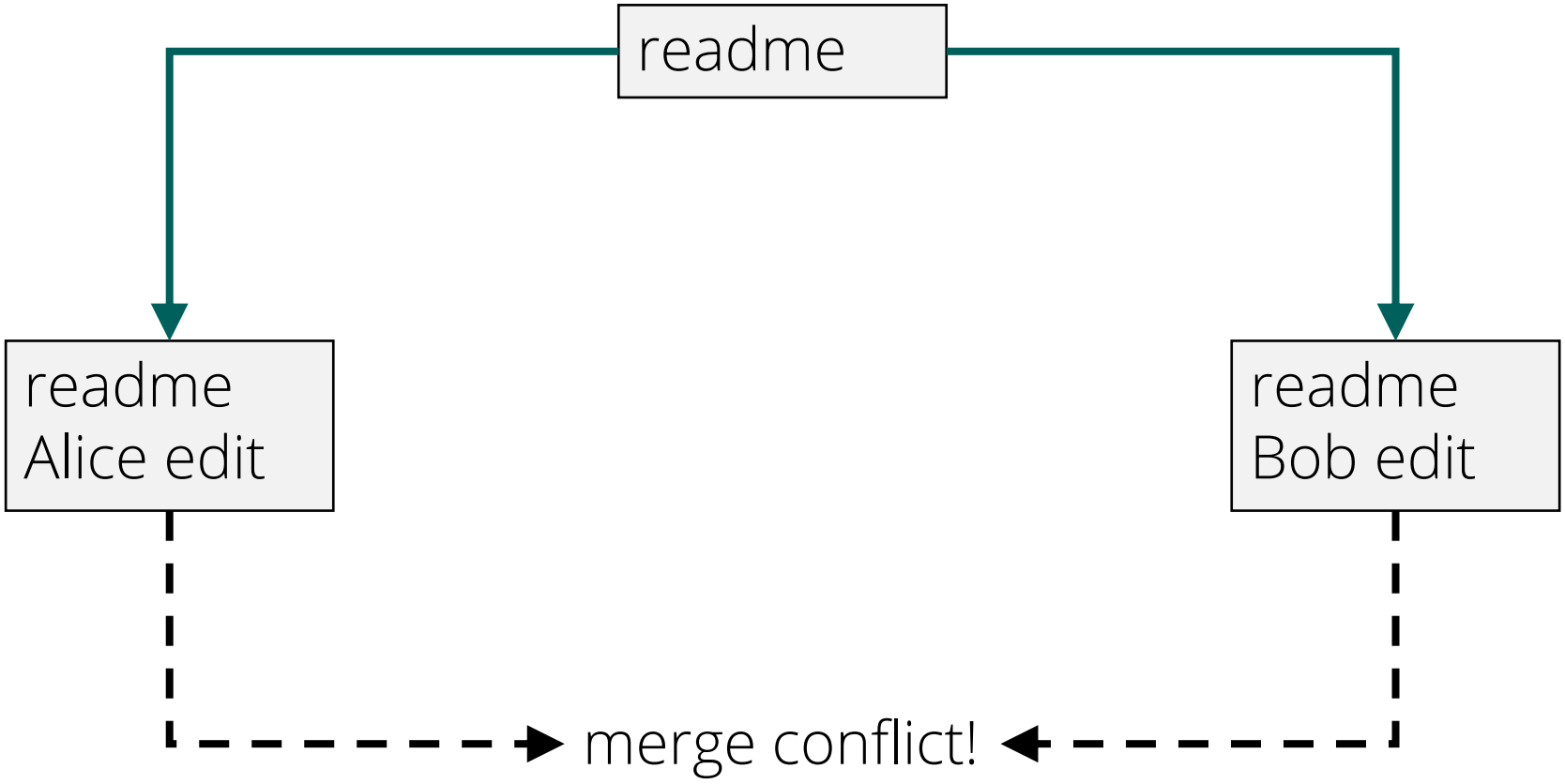


real conflicts

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

$ git pull
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
Automatic merge failed; fix conflicts and then commit the
result.
```


real conflicts



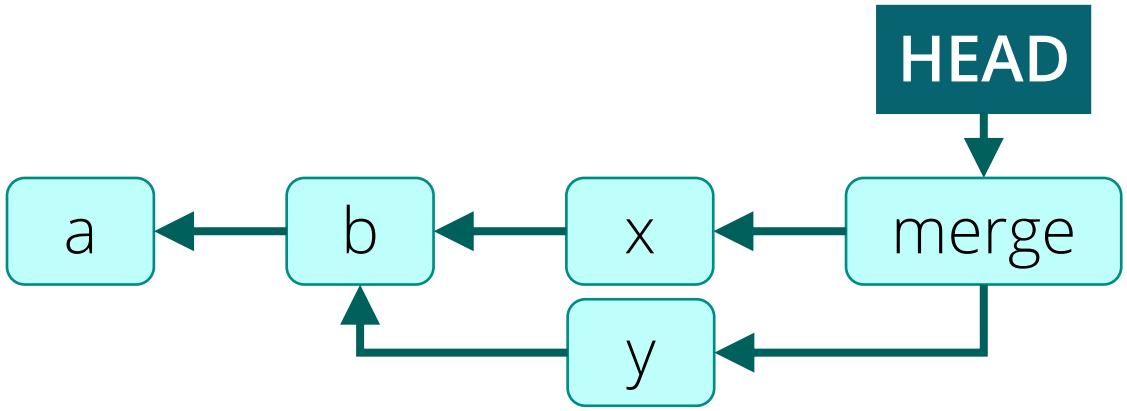
conflicts

```
bob$ cat README.txt
readme
<<<<<<< HEAD
Bob edit
=======
Alice edit
>>>>>>> fa63b5aa7e669a5d54034d6b956ab64240b3c251
```

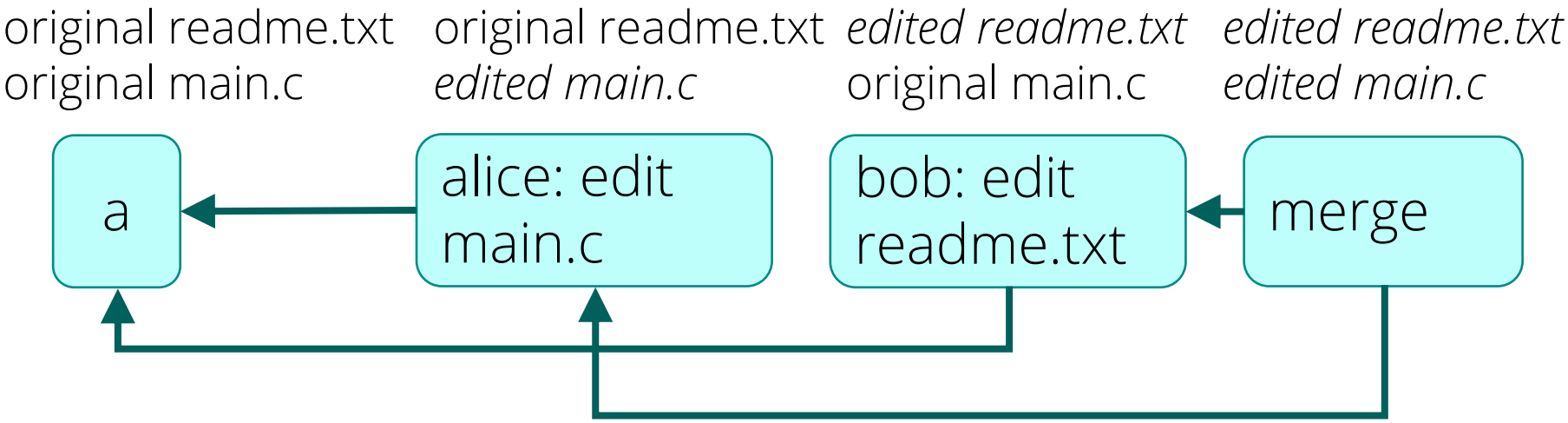

conflicts

1. fix all conflicts manually
2. make a commit

This creates a merge commit with more than one parent:



merge commits

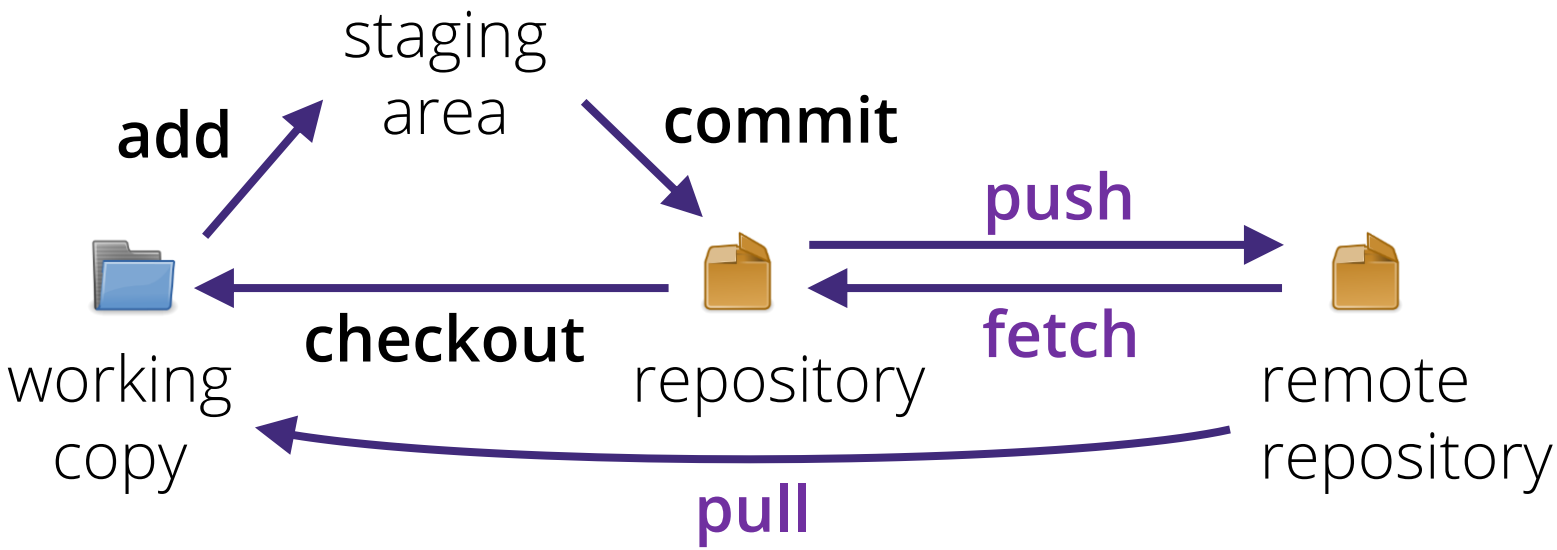


summary

- first thing in the morning: `fetch`, then `status`
- ahead: safe to `push`, behind: safe to `pull`
- in case of conflict: `pull`, (fast forward), merge and `commit` again

We can make even better workflows once we know about branches.

diagram



init **status** **clone** **merge**

log