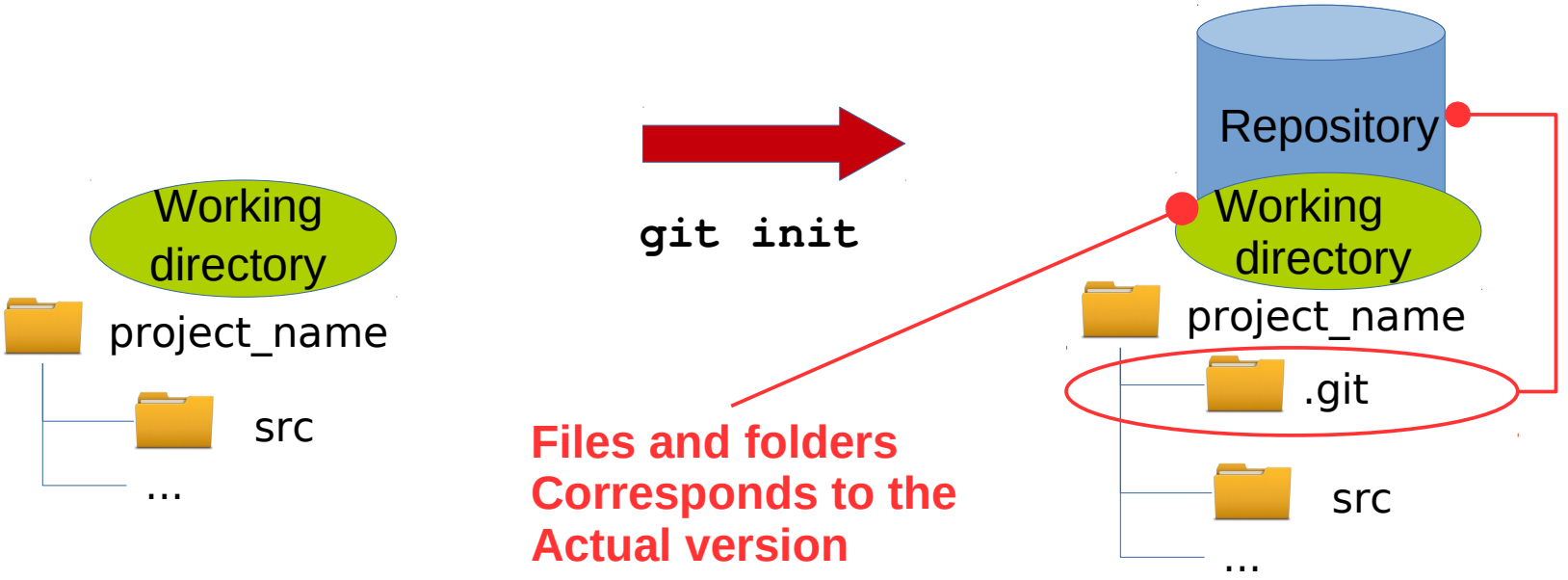
Concepts of GIT

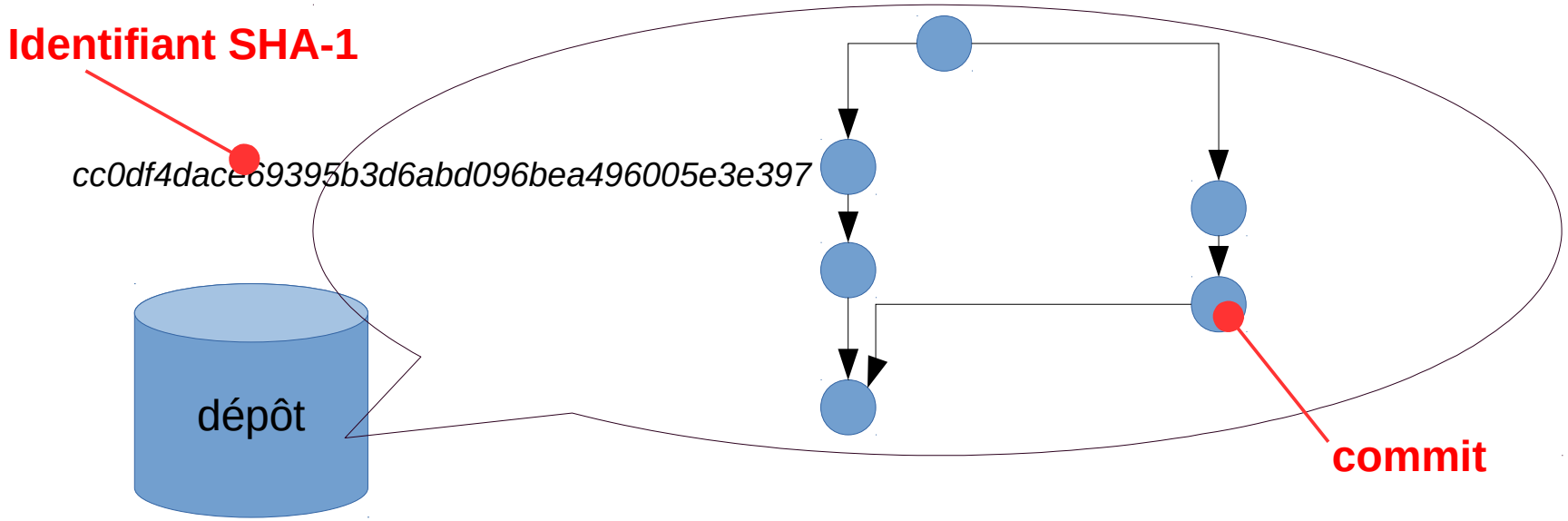
- **The repository**
  - _ A folder with all history of changes developpment Brancheetc.
  - _ Create a repository from a directory: **`git init`**



Concepts of GIT

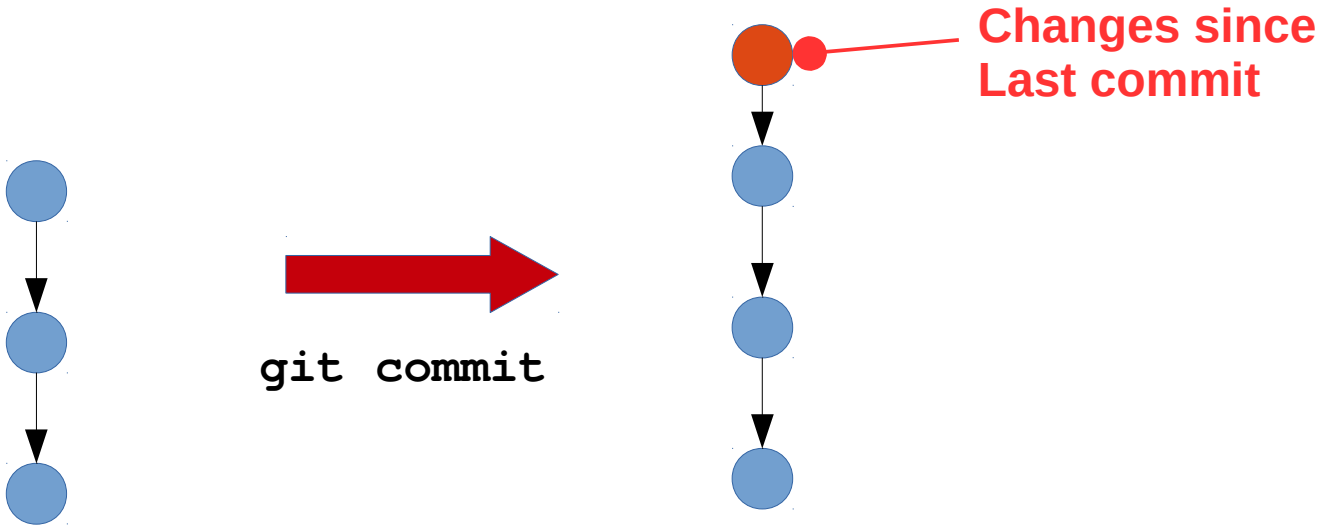
- The **commit**
  - A set of changes in dercotory and files
    - Represented by an operation of ***patch***.
  - Identified by a unique hash code SHA-1

- The commit are structured by an **acyclic oriented graph**



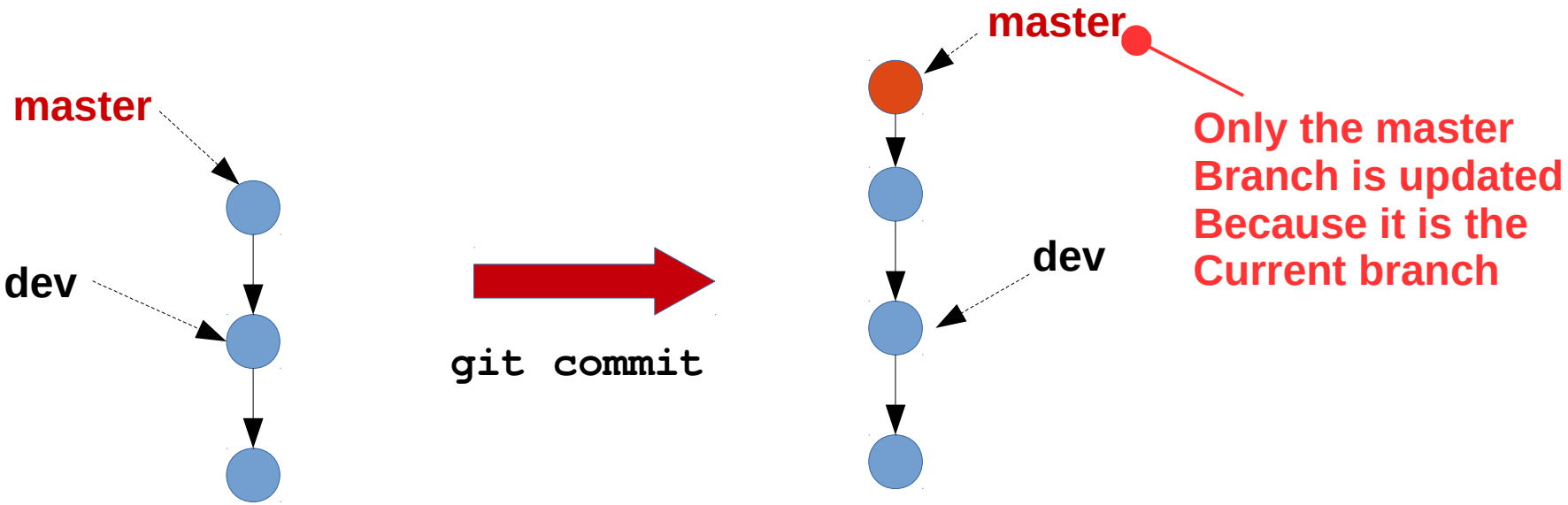
Concepts of GIT

- The **commit**
  - The graph is updated at each commit creation (e.g. action « *to commit* »)
    - The new commit points towards the preceding commit and It contains the changes since the previous commit



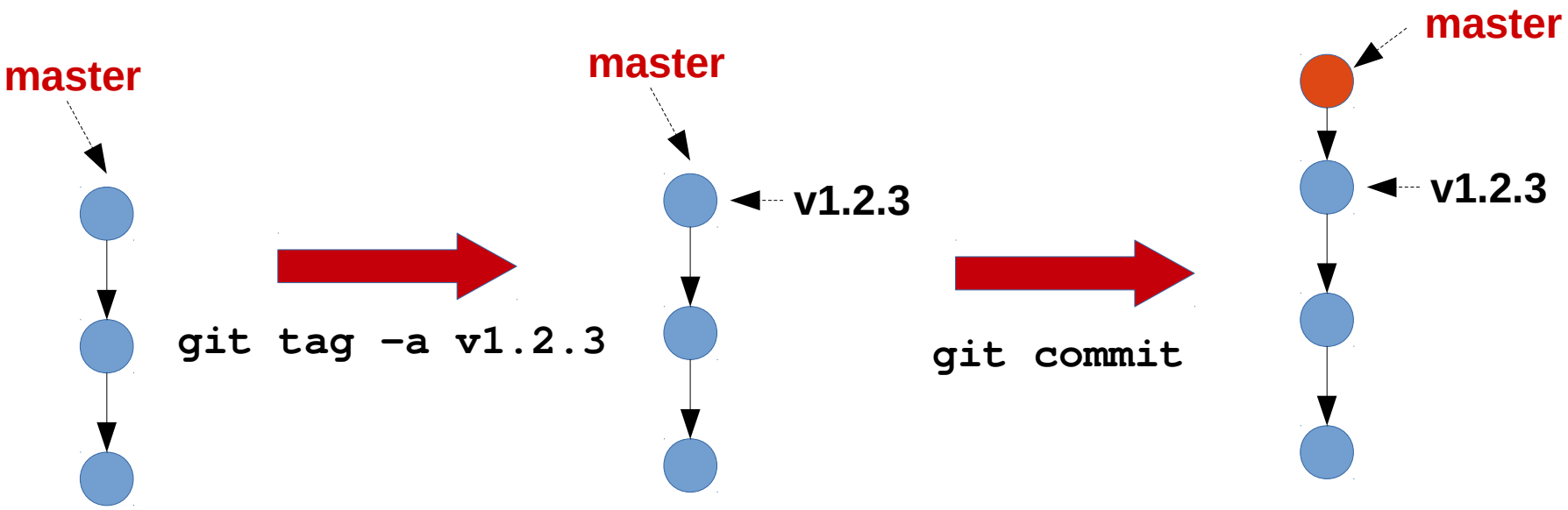
Concepts of GIT

- **The branch**
  - A **pointer that refers to** a commit and updated at Each new commit (then it refers to the new commit)
  - Default branch: **master**.
  - Git defines a **current branch**



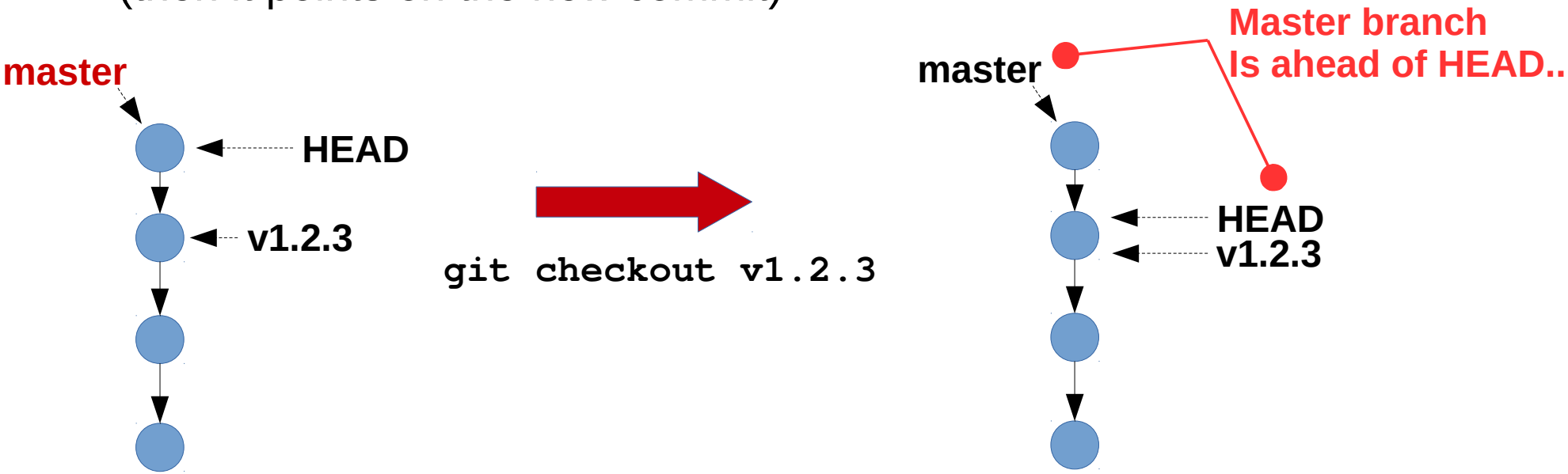
Concepts of GIT

- The **tag**
  - A pointer making reference to a commit but only updated When the user asks
  - Used to identify interesting versions



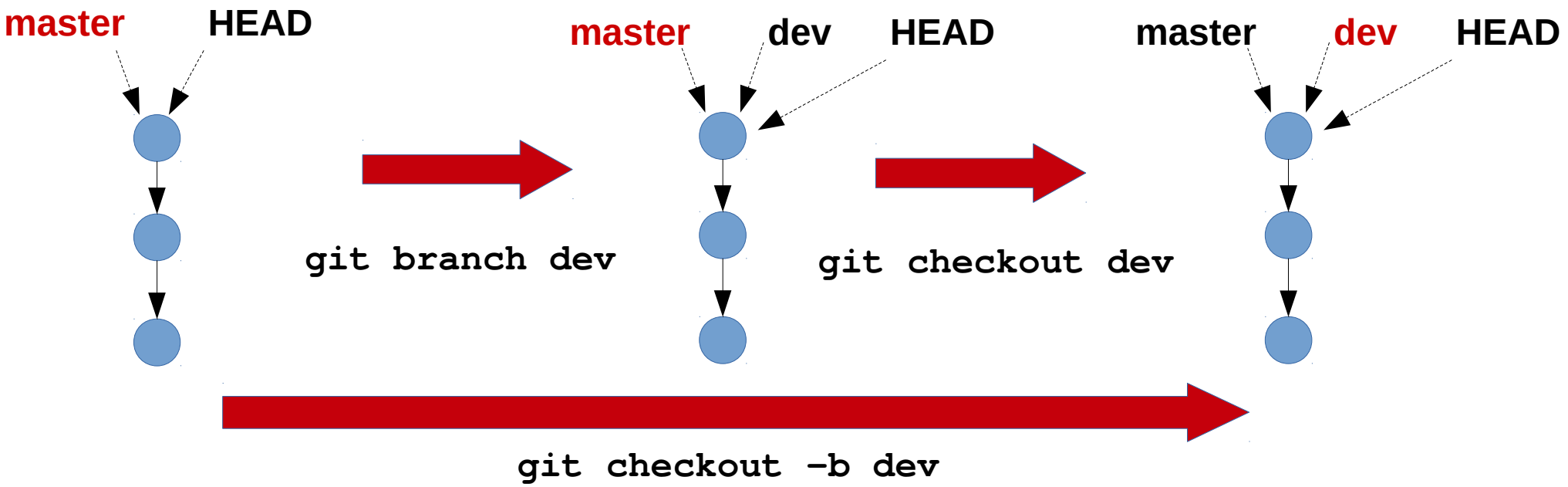
Concepts de GIT

- The **HEAD pointer**
  - A **pointeur** refering to a commit and representing the **current state of the Working directory** : all changes applied until the commit.
  - Used to navigate in the graph and explore the versions
  - Updated when the user asks but also each time a commit is created (then it points on the new commit)



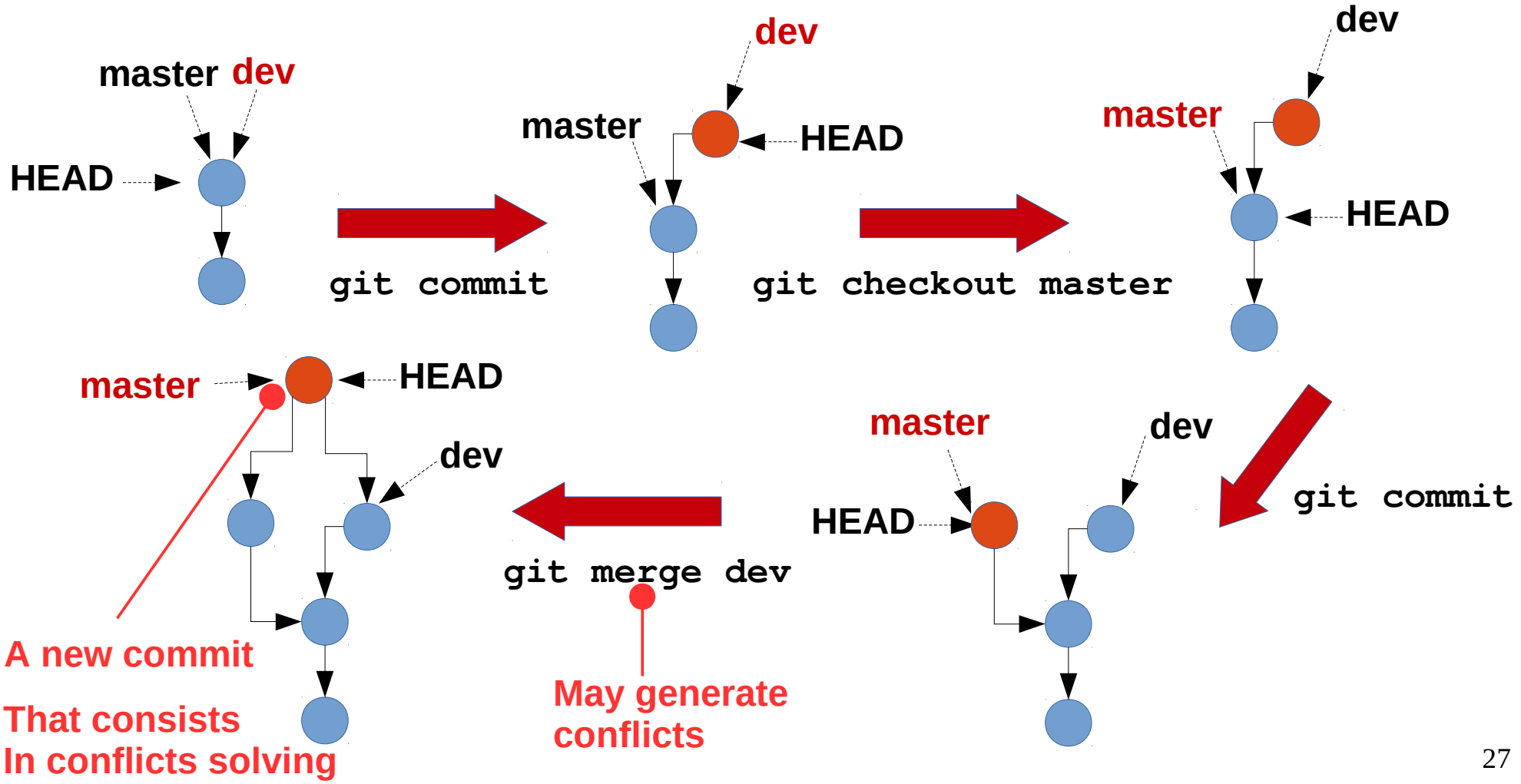
Concepts de GIT

- Create **branches**
  - There are several branches in the same repository
  - Create a branch = create and name a new pointer on the commit that point at the **HEAD**



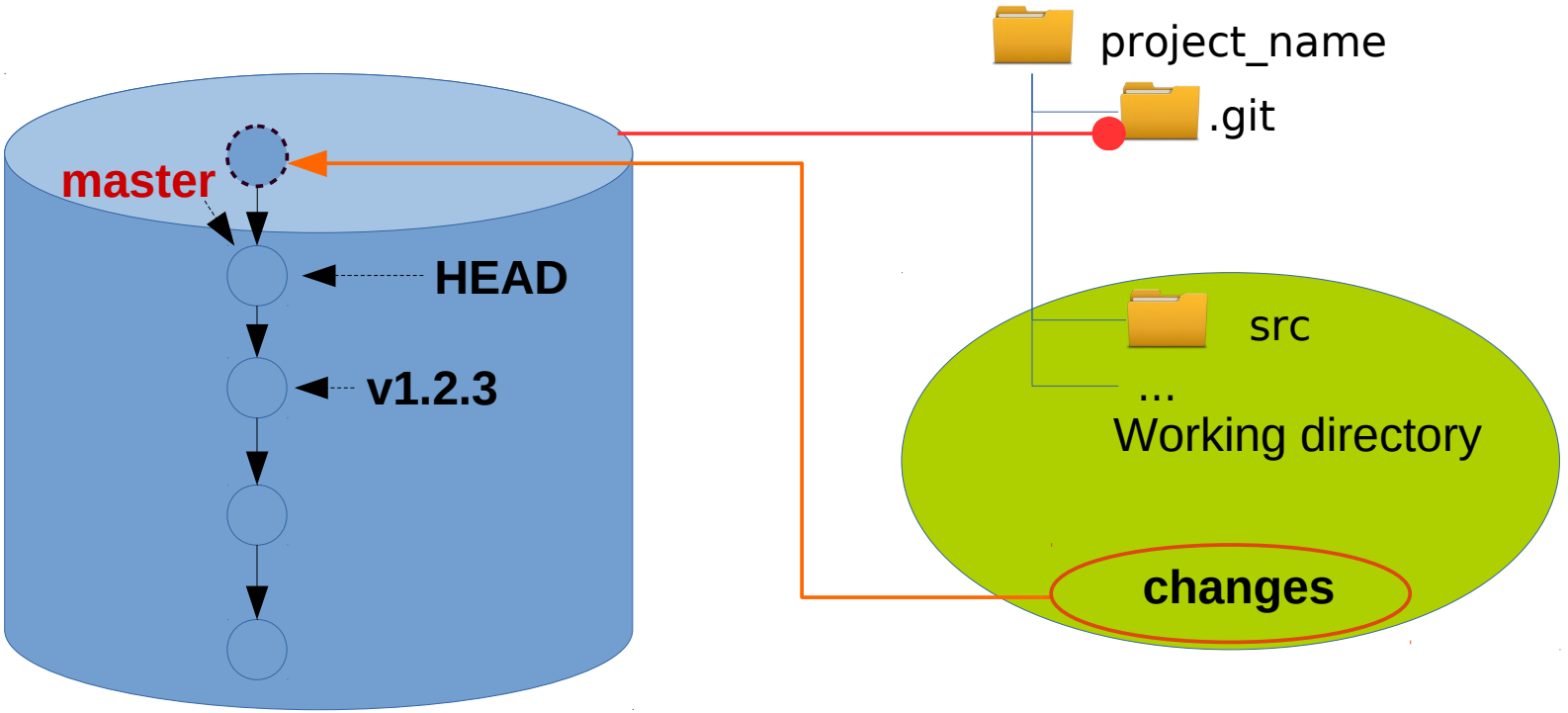
Concepts de GIT

- ***Merging*** branches



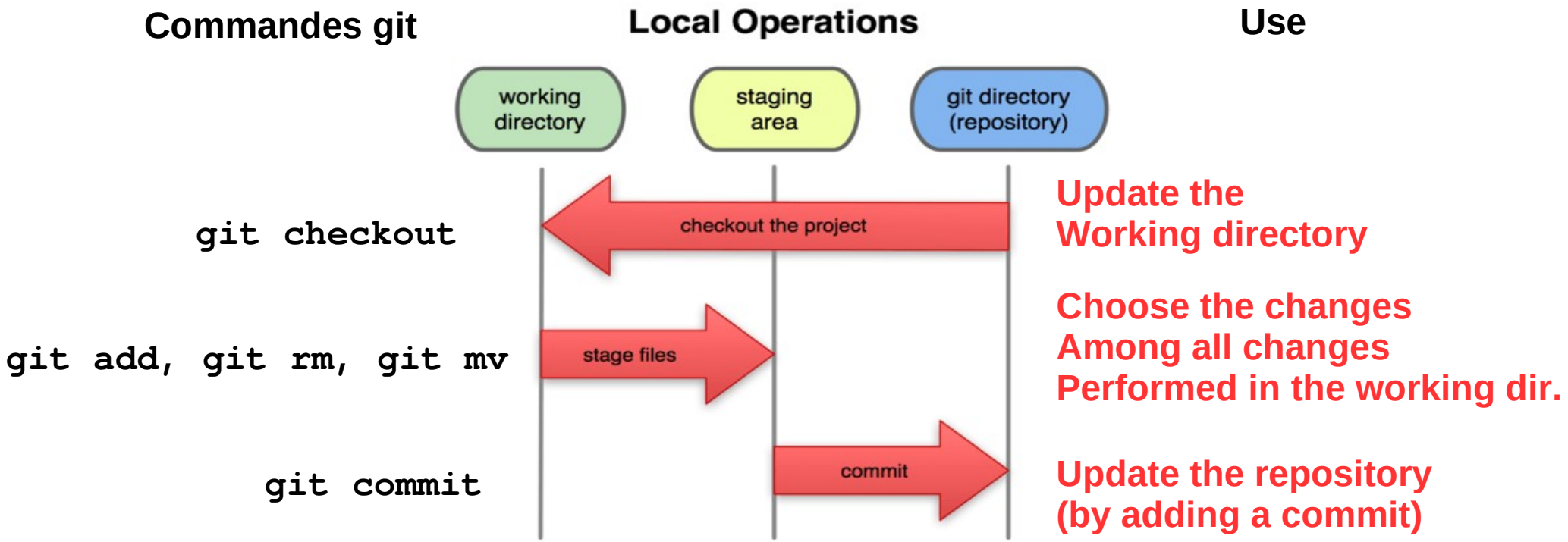
Concepts de GIT

- Create a commit from your changes
  - Goal : Save the changes between the content of The working directory (files and folders) and the HEAD (last stae saved)



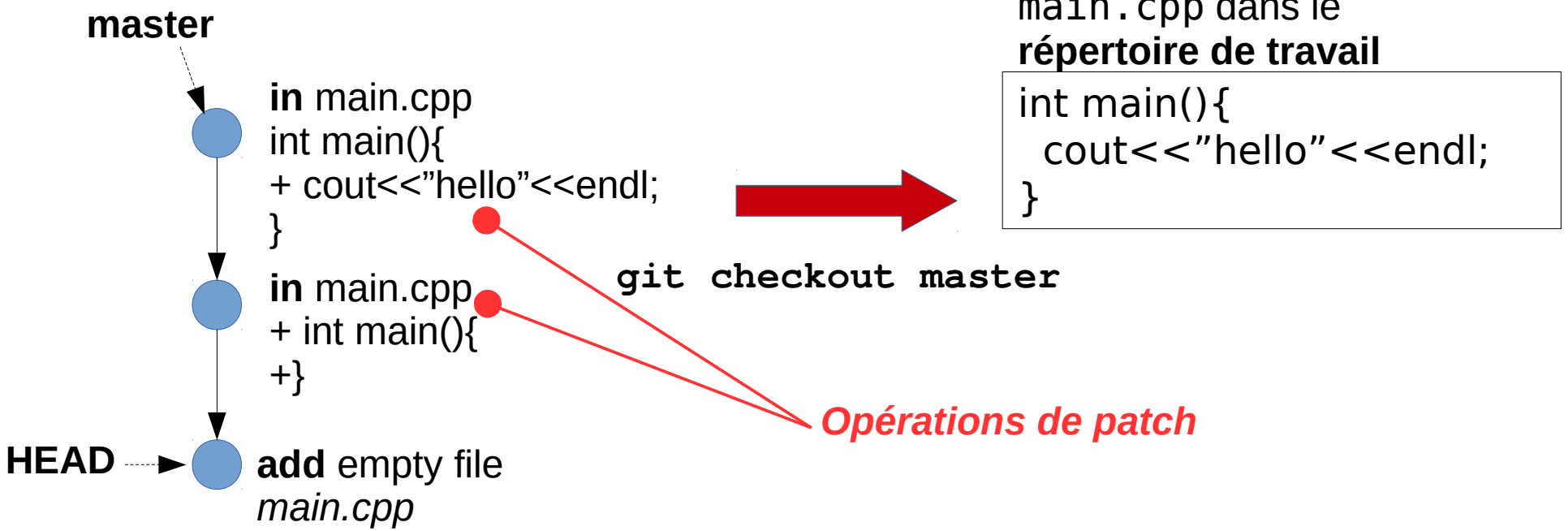
Concepts de GIT

- Create a commit from changes
  - Originality from git: 2 phases



Concepts de GIT

- Updating the working directory
  - At each `checkout` or `merge`
  - Applies the sequence of all changes (patches) fom the first current commit (pointed by HEAD)



Concepts de GIT

- Saving specific changes of the working directory
  - Select the changes that will be saved in the next commit
    - internes to files (use git add -p (never used) ; toutes les modification
      All the changes or a new file **git add filename**
    - Al changes (use git add --all
  - Unselect changes : **git reset**

main.cpp en **HEAD** dans le **dépôt**

```
int main(){
  cout<<"hello"<<endl;
}
```

main.cpp dans **répertoire de travail**

```
int main(){
  cout<<"hello"<<endl;
  return 0;
}
```

`git add main.cpp`

Changes saved
In the **staging area**

```
in main.cpp

cout<<"hello"<<endl;
+ return 0;
}
```

***diff* operation deduct the patch.**

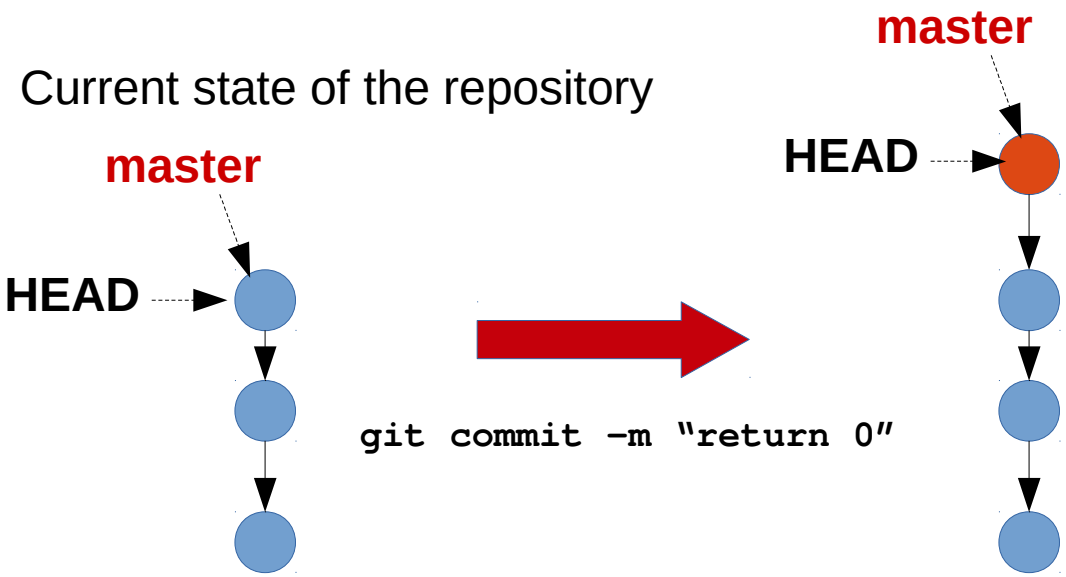
Concepts de GIT

- Update the local repository (« to commit »)
  - Create a new commit from the changes saved in the « *staging area* ».
  - Add a message that explains what has been changed

All changes saved in the **staging area**

```
add other.cpp
in main.cpp
 in main.cpp

 cout<<"hello"<<endl;
}+ return 0;
 }
```



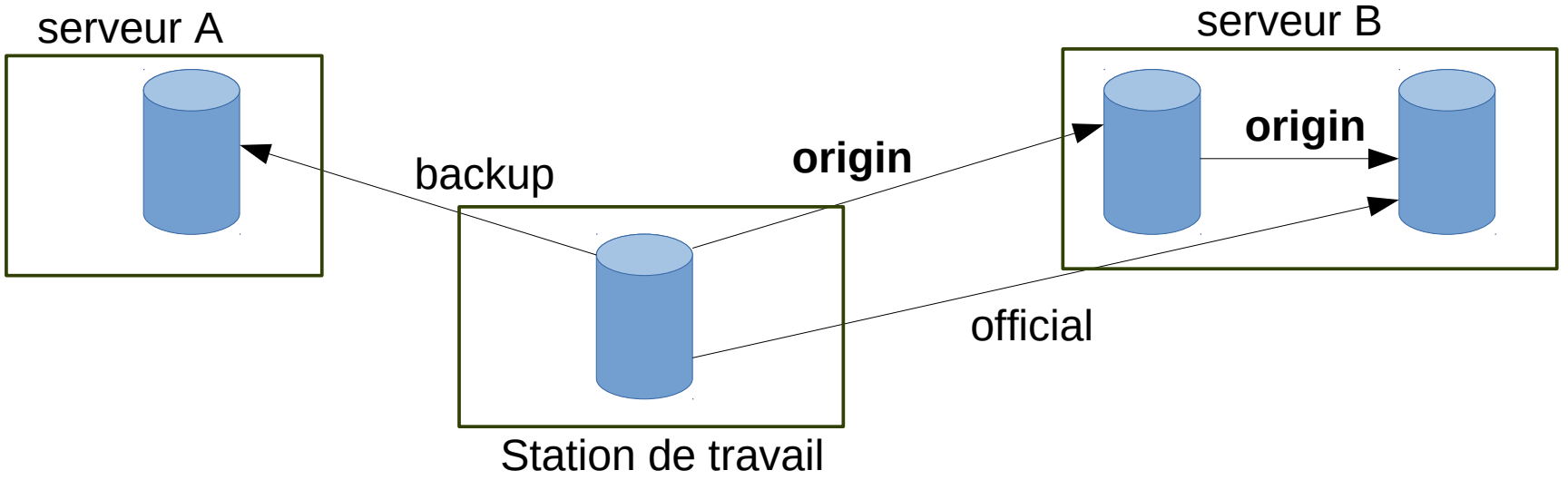
Concepts de GIT

- Pourquoi 2 phases pour créer des commits ?
  - créer de « petit commits » à partir d'un grand nombre de modifications.
  - retarder le commit de certains modifications: les fonctionnalités validées peuvent être « commitées » dessuite.
  - isoler le code qui ne sera jamais commité (e.g. les traces de debug).

- Une fois que les modifications intéressantes sont « commitées », nettoyer la staging area et répertoire de travail est possible :
  - Utiliser `git reset --hard` : suppression définitives des modifications non commitées.
  - Utiliser `git stash save` : modifications non commitées peuvent être restaurées.

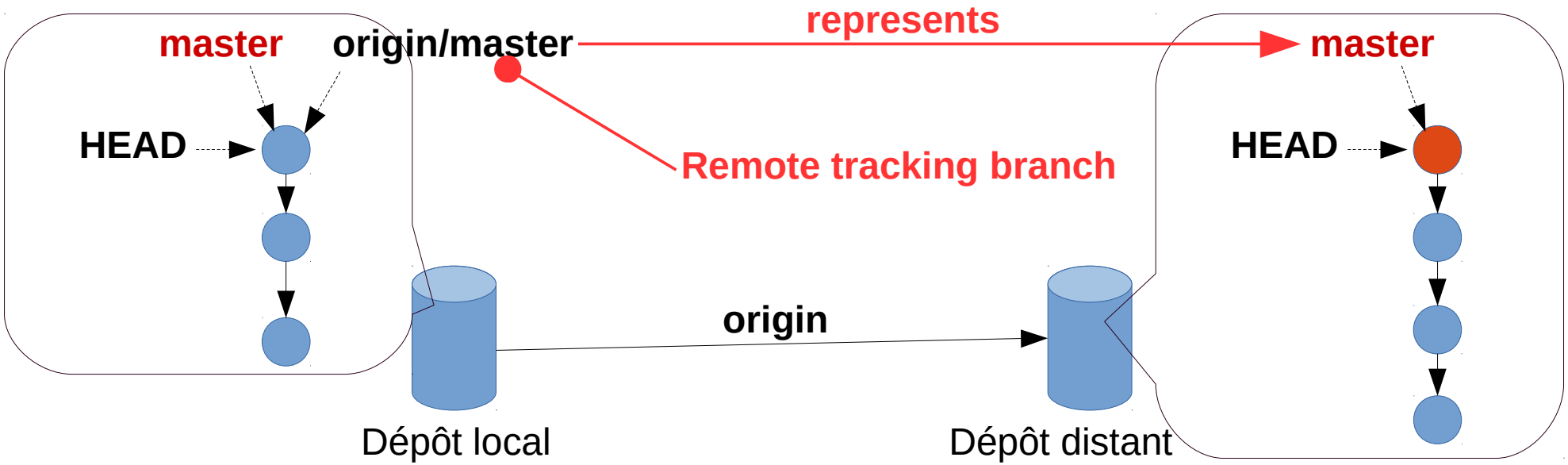
Concepts de GIT

- *Syncronisation with remote repositories*
  - Each repository knows a set of ***remotes***.
  - The default *remote* is called **origin**.
  - Each *remote* is associated to a unique **name** (unique in the local repos) and **Defines an url** (The adress of the remote).
  - Add a *remote* : `git remote add backup <address>`
  - remove a *remote* : `git remote rm backup <address>`



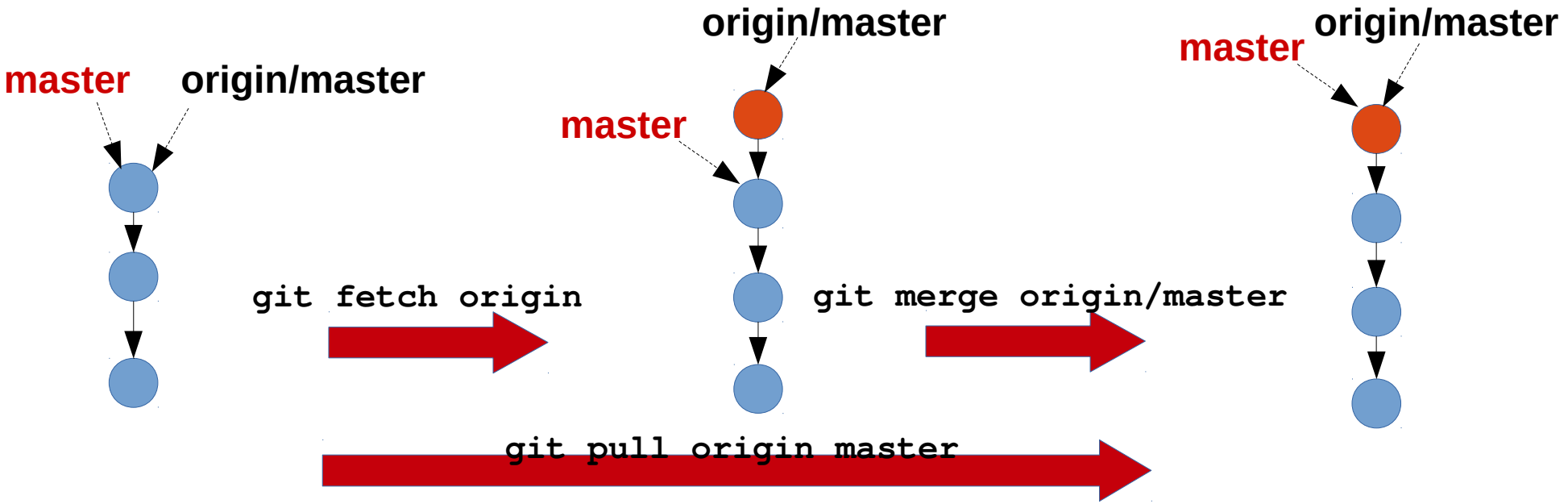
Concepts de GIT

- Synchronizin **local branches with remote branches**
  - Branches of *remotes* are available on the local repository **BUT** They are on **read only (you cant create commit on them)** We call them ***remote tracking branches***



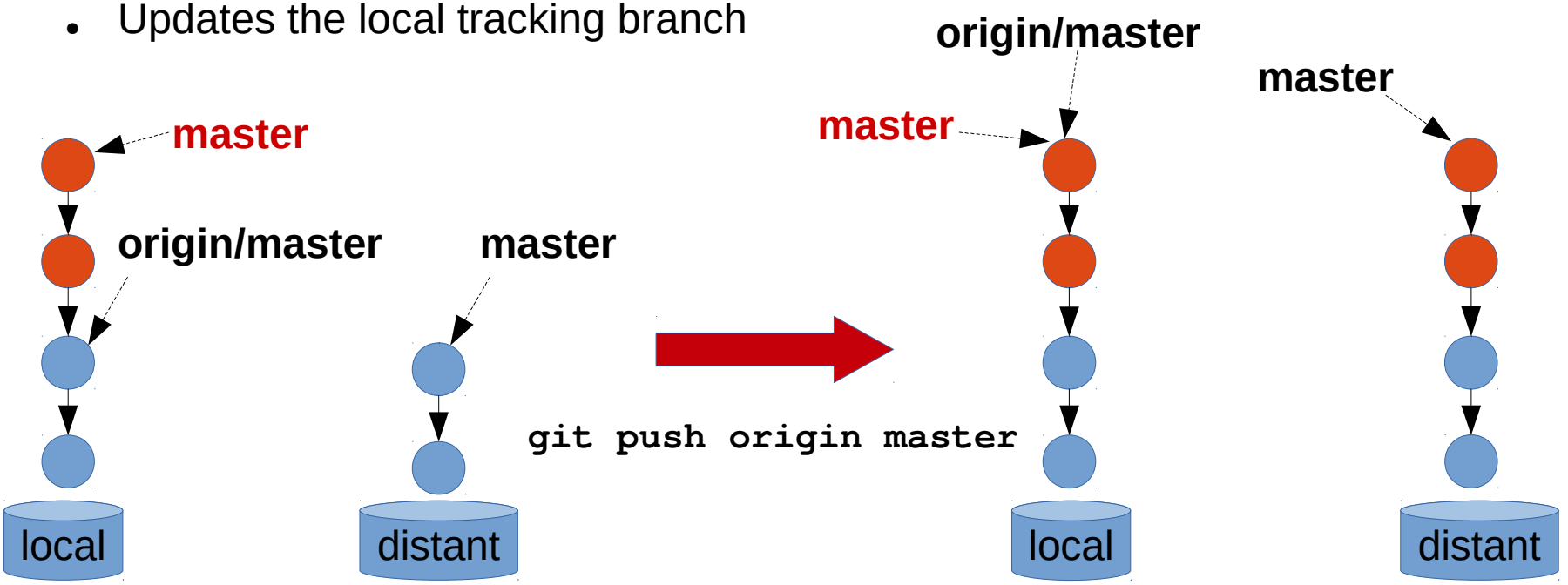
Concepts de GIT

- Update your local repository from the *remotes*.
  - Update (if needed) the graph of **local commits** and the **Remote tracking branches** (fetch command)
  - Merging the granch of local changes and the remote tracking branches (commande `merge`).
  - Do both in one: `pull`



Concepts de GIT

- Update a remote branch from a local branch
  - Use **push** command (atomic operation)
    - Checks that the local branch is up to date
    - Update the commit graphs of the remote and the corresponding branch
    - Updates the local tracking branch



Concepts de GIT

- **Clone repository**
  - Create the local directory and set it as a repository (git init).
  - Create *remote* called ***origin*** (git remote add origin <address>)
  - Create a local ***master*** branch (git checkout -b master)
  - Updates the ***master*** local branch (git pull origin master).

Adresse: login@server:repo.git

Concepts de GIT

- Typical use

```
git pull/fetch/merge ...

git add/rm/mv/reset ...

git commit ...

git push ...
```