Cours 6 : Commandes utiles.

Le stash.

Le stash permet de mettre de côté des modifications.

Git a besoin d'avoir une *copie de travail propre* pour pouvoir faire certaines opérations.

```
Please, commit your changes or stash them before you can merge.
Aborting
```

Stasher permet de sauvegarder notre travail pour pouvoir le réutiliser après.

Appliquer un stash[1].

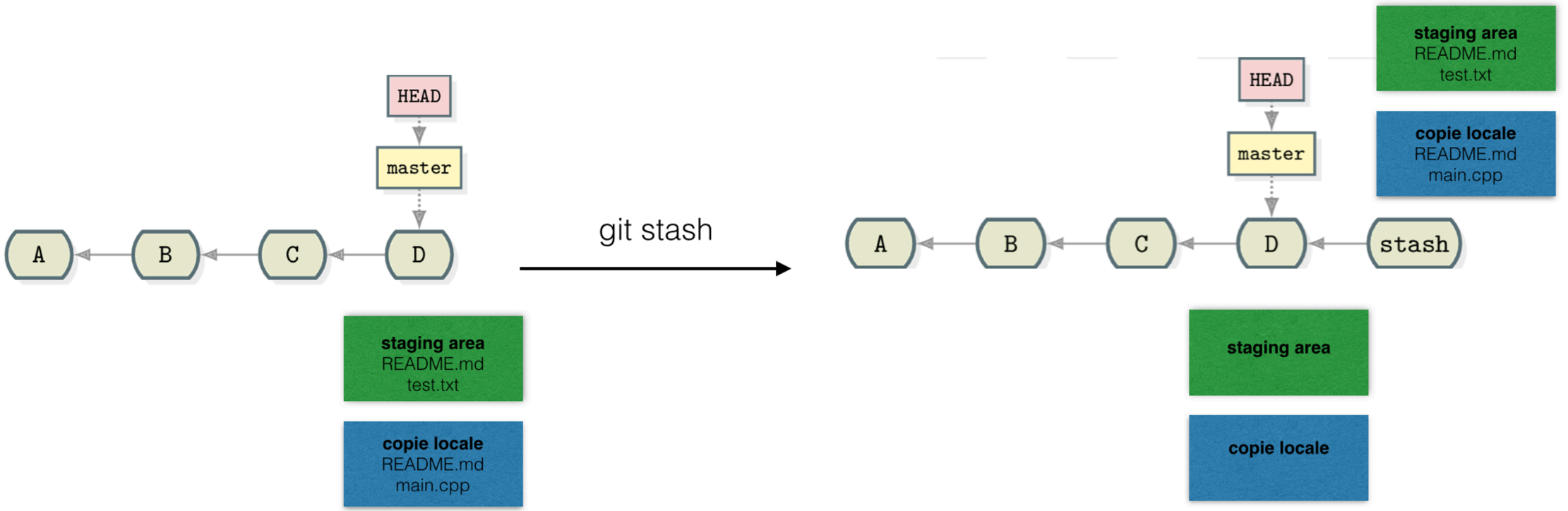
```
1 | git stash [save ["message"]]
```

Par défaut, git prendra le message du dernier commit :

```
➜  Realtime-Markdown-Viewer git:(code) ✗ git stash
Saved working directory and index state WIP on code: db9e80b add code in parser to display several
 lines of code
HEAD is now at db9e80b add code in parser to display several lines of code
```

[1] Peut être source de conflits.

Illustration.



Fonctionnement du stash.

Le stash est commun à toutes les branches et fonctionne sur le principe d'une liste[2].

```
1 | git stash list
```

```
stash@{0}: WIP on code: db9e80b add code in parser to display several lines of code
stash@{1}: WIP on new_line: 7630886 add code in parser
stash@{2}: WIP on tp5: b496d8c Revert "frf"
stash@{3}: WIP on tp4: b496d8c Revert "frf"
(END)
```

[2] Cette liste reste locale.

Réappliquer une modification.

Le pop enlevera le stash de la liste :

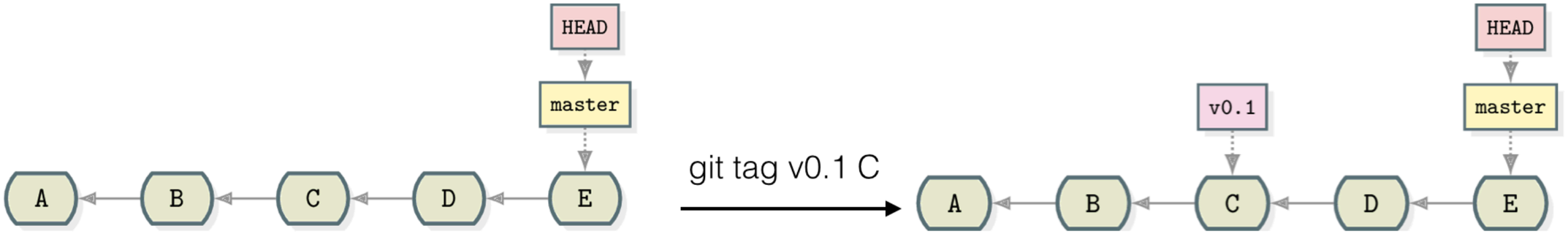
```
1 | git stash pop [--index] [stash@{N}]
2 | git stash apply [--index] [stash@{N}]
```

Pour visualiser le contenu du stash :

```
1 | git show stash@{N}
```

Les tags.

Un tag est une référence **fixe** vers un commit.



Git possède deux types de tags : les tags légers et les tags lourds.

Créer un tag.

Pour créer un tag léger :

```
1 | git tag TAG [REF]
```

Pour créer un tag lourd[3] :

```
1 | git tag -a [-m MESSAGE] TAG [REF]
```

[3] Un tag lourd est un objet git. On y retrouve les informations d'un commit : date, auteur ...

Utiliser un tag :

```
1 | git checkout [-b NEWBRANCH] TAG
```

Pour partager un tag[4] :

```
1 | git push --tags
2 | git push REMOTE TAG
```

[4] Attention par défaut le push ne partage pas les tags.

Supprimer un tag.

Pour supprimer un tag localement :

```
1 | git tag -d TAG
```

Pour supprimer un tag à distance :

```
1 | git push --delete REMOTE TAG
```

Gitignore.

Ignorer des fichiers.

Ignorer des fichiers pour ne pas qu'ils apparaissent dans le `git status`.

Fichier à la racine du projet[5] : **.gitignore** [6]

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so
.gitignore (END)
```

[5] Peut être aussi défini dans chaque sous répertoire.

[6] Pour des exemples type : ici

Le reflog.

Retrouvez les orphelins[7].

```
1 | git reflog
```

Chaque opération de git est enregistrée :

```
b496d8c HEAD@{0}: merge tp5: Fast-forward
c853da9 HEAD@{1}: checkout: moving from master to tp4
1a4c660 HEAD@{2}: checkout: moving from c3af8efe2bd837f0994ad8b0c6315d514b2dbe0e to master
c3af8ef HEAD@{3}: checkout: moving from master to c3af8ef
1a4c660 HEAD@{4}: checkout: moving from tp5 to master
b496d8c HEAD@{5}: checkout: moving from master to tp5
1a4c660 HEAD@{6}: reset: moving to origin/master
5d78b73 HEAD@{7}: reset: moving to HEAD^
1a4c660 HEAD@{8}: checkout: moving from tp5 to master
b496d8c HEAD@{9}: revert: Revert "frf"
0baab10 HEAD@{10}: commit: frf
c853da9 HEAD@{11}: checkout: moving from code to tp5
db9e80b HEAD@{12}: commit (amend): add code in parser to display several lines of code
```

[7] Git stocke les orphelins pendant 90 jours, après ils sont supprimés via un garbage collector.