GIT-2 of 4

SIMANTA MITRA

MAIN REFERENCE:

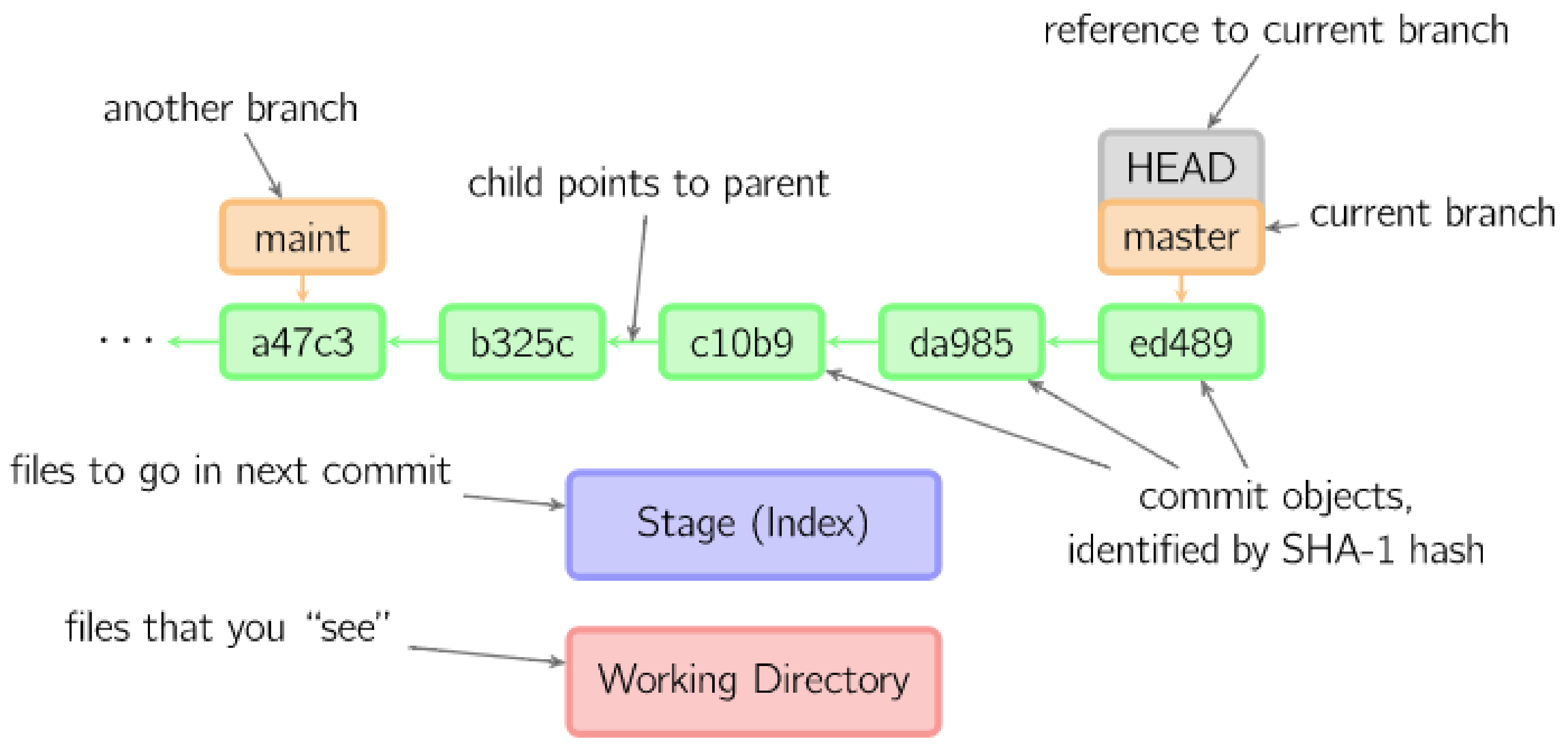
HTTP://MARKLODATO.GITHUB.IO/VISUAL-GIT-GUIDE/INDEX-EN.HTML

Learning Objectives

Now that you have some basic idea of GIT, lets learn about commands that you will use when working with your local repository i.e. add, commit, checkout, reset, and diff.

Operation	Developer wants to	Command
SAVE	stage files	git add
SAVE	create a snapshot or commit object	git commit
RETRIEVE	get file from stage	git checkout -- filename
RETRIEVE	get file from snapshot	git checkout
UNDO	unstage files	git reset
UNDO	undo a commit	git reset HEAD~1
UNDO	overwrite a commit	git commit --amend
DIFFERENCE	find difference between versions of files	git diff

Also, you will need to know about HEAD and master and disambiguation.



HEAD is a pointer. It points to the current branch. master is NAME of the default branch.

Remember that history is a linked list of COMMIT objects. New commit objects point to parent commit objects.

disambiguation option

- In many GIT commands we refer to FILES as well as BRANCHes. What happens if file name **is same as a** branch name?
- To avoid such issues, in GIT commands

  -- NAME means filename is being used

  NAME -- means branch name is being used
- If there are no clashes, -- can be omitted.

- Example:

  git reset -- XYZ **(we mean file name here)**

  git reset XYZ -- **(we mean branch name here)**

Do note that we use the word commit

- as a verb (git operation of commit)
- as a noun (commit object)

save to stage

ADD

SAVE to STAGE

add

This command adds files FROM working directory to the stage.

- You can add specific files ex: **git add abc.txt**
- You can add entire folder recursively ex: **git add .**

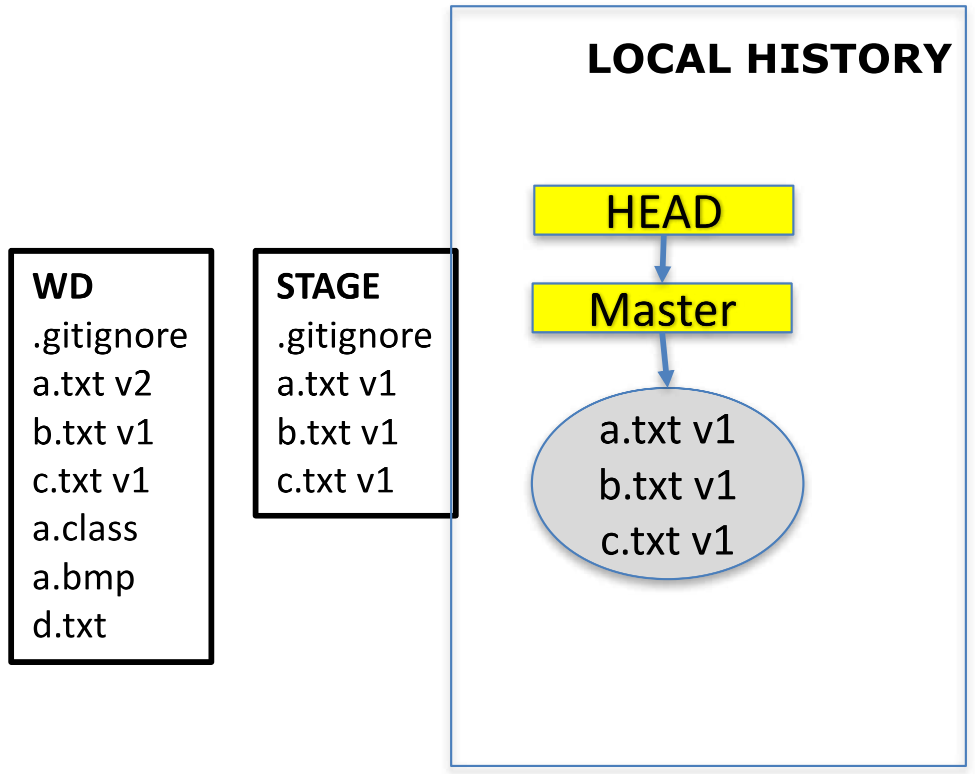
Remember that files indicated in .gitignore will not be added.

Remember that you are REPLACING existing files in the stage with the new versions.

SAVE to STAGE

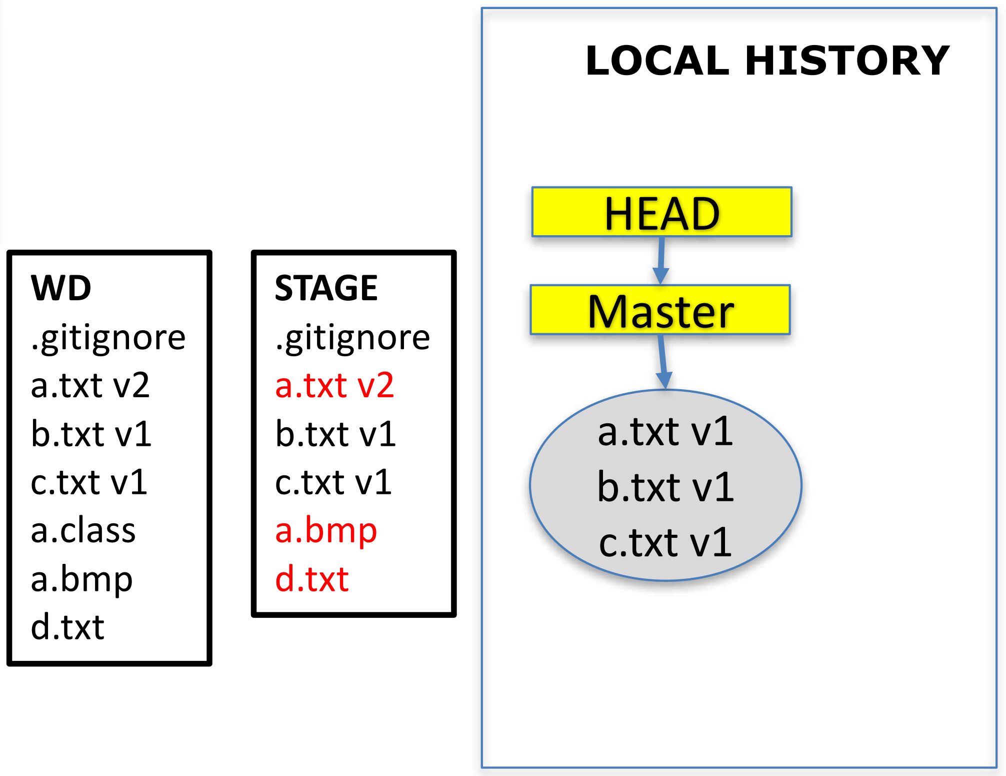
git add .

BEFORE ADD OPERATION



Lets assume that .gitignore file has *.class in it.

AFTER ADD OPERATION



Note that a.class did not get added to stage

Note that new version of a.txt is in stage now

Note that history did not change

save to history

COMMIT

SAVE to HISTORY

commit

This command:

1. creates a new COMMIT object in history with files from **stage**
2. makes this commit object point to the current commit object.
3. points the current branch to this new commit object.

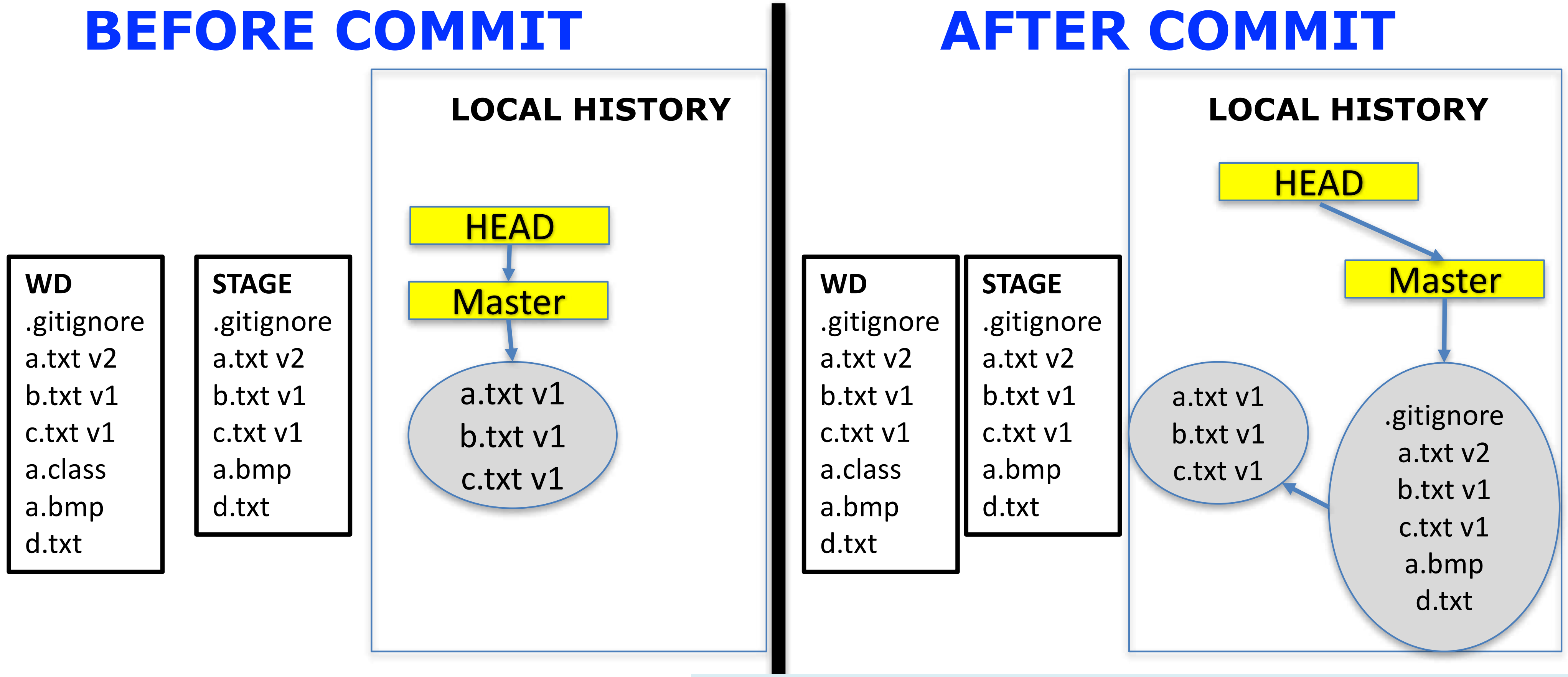
You have to give a comment along with the commit. Make sure it is meaningful.

You can combine add and commit (if a file is already being tracked) using –a option.

example: **git commit –a abc.txt**

SAVE to HISTORY

git commit



1. from files in stage a new commit object got created in history
2. this points to old commit object
3. head and branch points to new commit object

Note that WD & stage did not change.

SAVE to HISTORY

commit --amend

sometimes, we do a commit a bit prematurely. Say we want to fix a bug. We think it is fixed and commit.

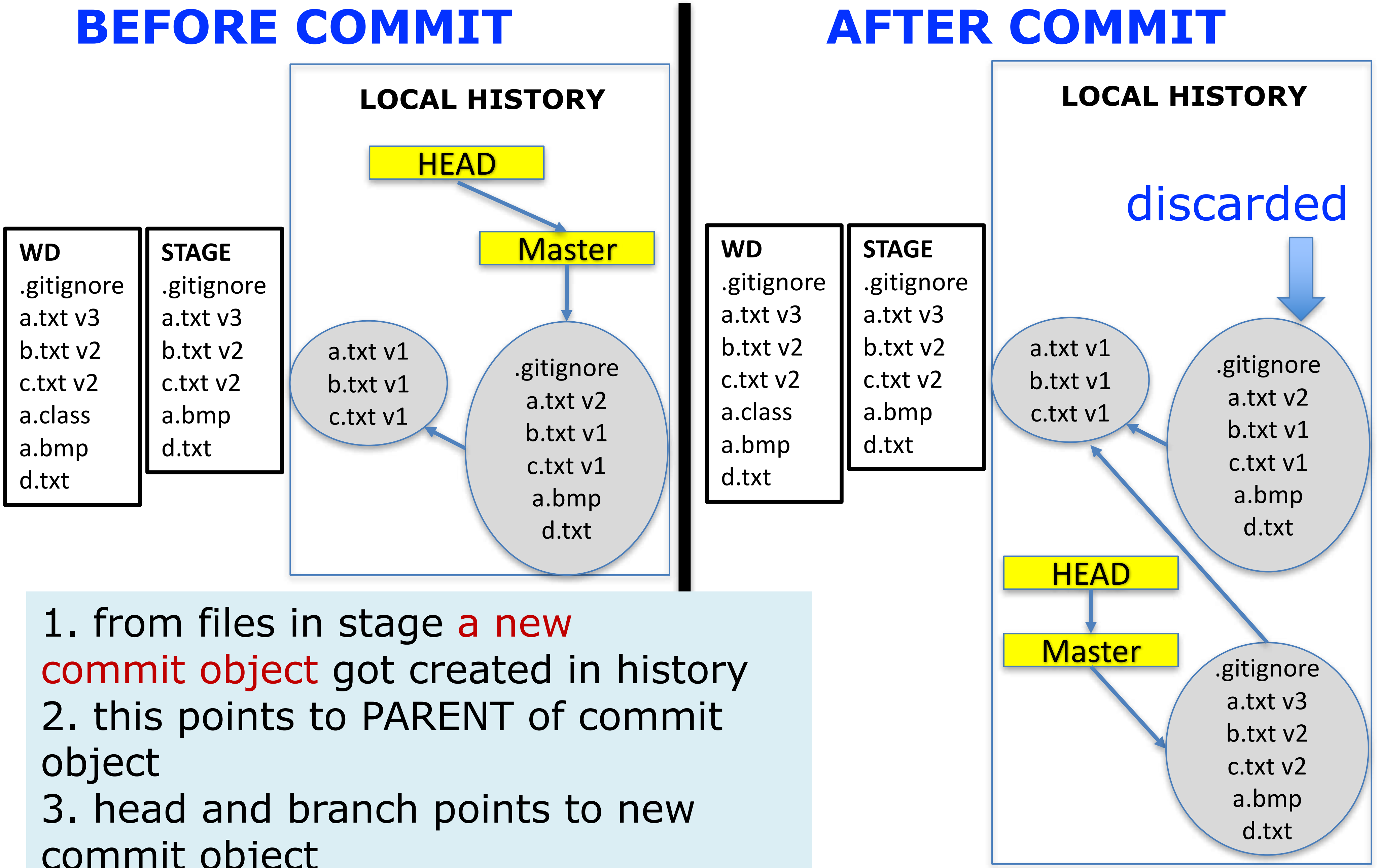
Then, we find out we need to make more changes. The intermediate commit was not useful.

One way to get rid of it is by using commit --amend option. This will create a NEW commit but make it point to the parent of the intermediate commit.

Thus, the intermediate commit will be discarded as nothing points to it).

SAVE to HISTORY

git commit --amend



1. from files in stage a new commit object got created in history
2. this points to PARENT of commit object
3. head and branch points to new commit object
Note that WD & stage did not change.

retrieve from history or stage

CHECKOUT

RETRIEVE

checkout

This command has two versions.

- **Version 1:** git checkout COMMIT_OR_BRANCHNAME

This version retrieves from HISTORY and overwrites both STAGE and WD. Also, moves HEAD pointer (*Optionally, you can indicate FILENAMEs after BRANCHNAME to retrieve only indicated files).*

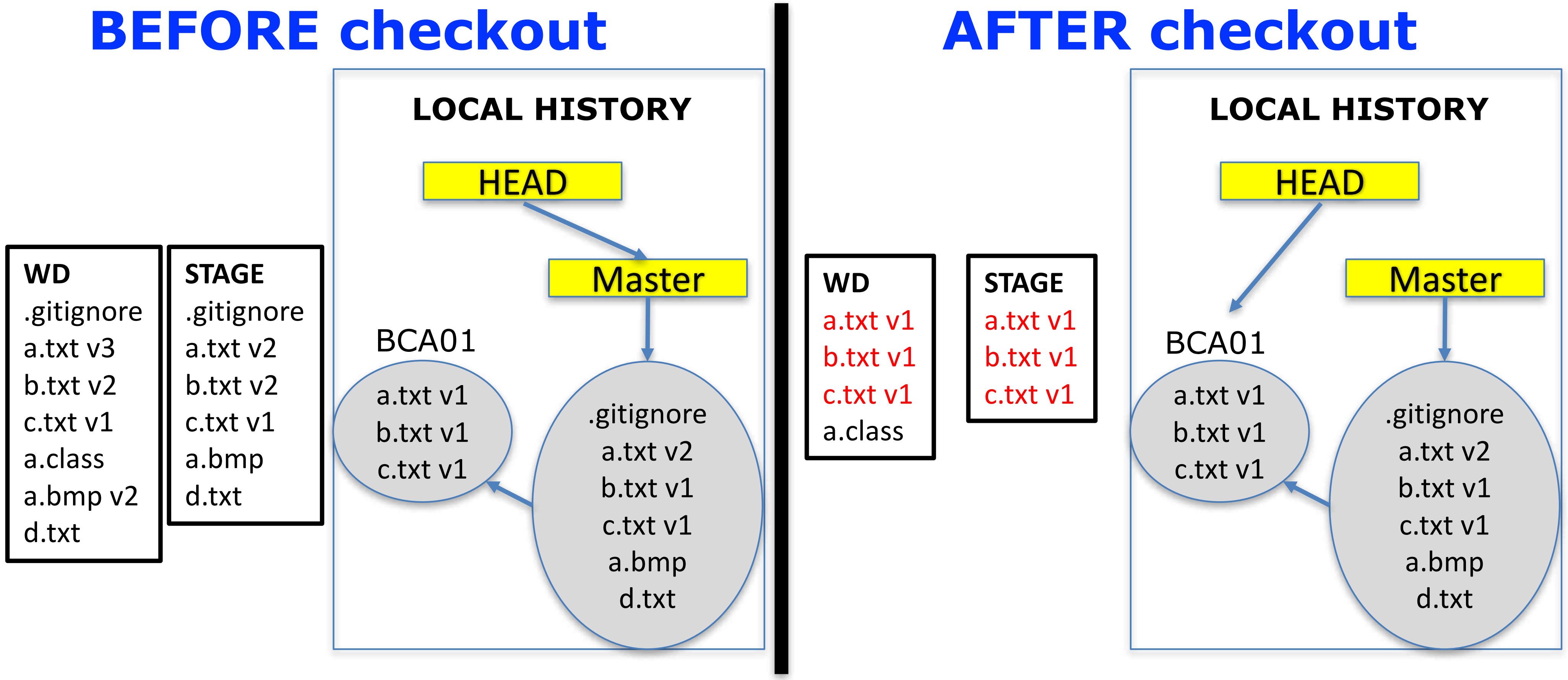
- **Version 2:** git checkout FILENAME (can use -- for disambiguation when needed)

This version retrieves from STAGE and overwrites WD

(throws away changes to file in WD)

RETRIEVE

git checkout VERSION1



git checkout BCA01

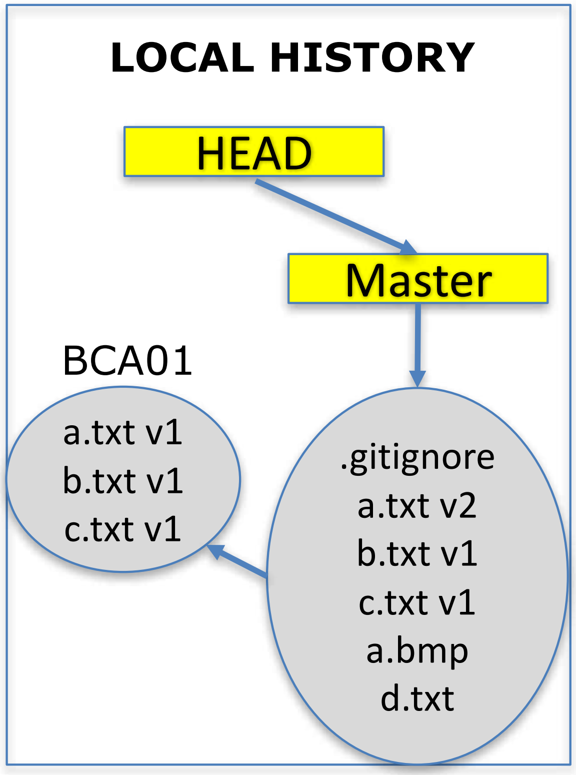
1. note that files are copied from commit object BCA01
2. note that UNTRACKED files are left alone (like a.class)
3. note that WD and STAGE are overwritten
4. note that HEAD changed.

RETRIEVE

git checkout VERSION1

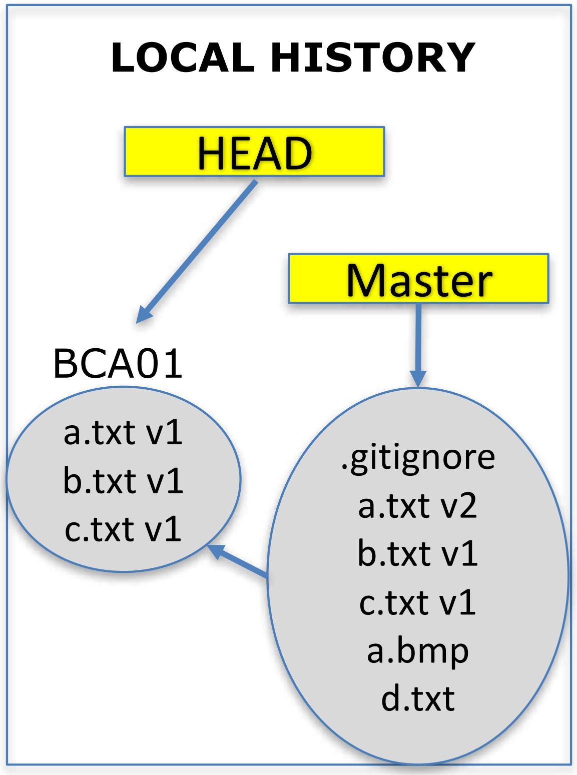
BEFORE checkout

WD	STAGE
.gitignore	.gitignore
a.txt v3	a.txt v2
b.txt v2	b.txt v2
c.txt v1	c.txt v1
a.class	a.bmp
a.bmp v2	d.txt
d.txt	



AFTER checkout

WD	STAGE
a.txt v1	a.txt v1
b.txt v1	b.txt v1
c.txt v1	c.txt v1
a.class	



git checkout HEAD~1

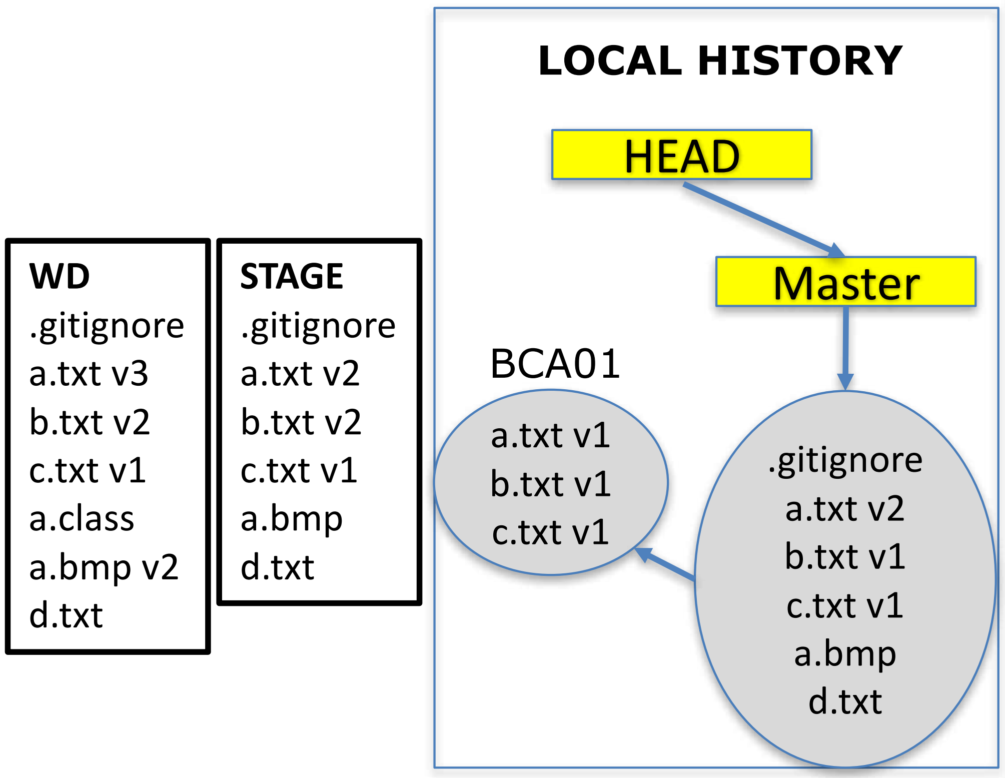
~ notation is used to indicate previous.
HEAD~1 means parent of node pointed by HEAD
HEAD~2 means parent of HEAD~1 etc

In this example, HEAD~1 means BCA01

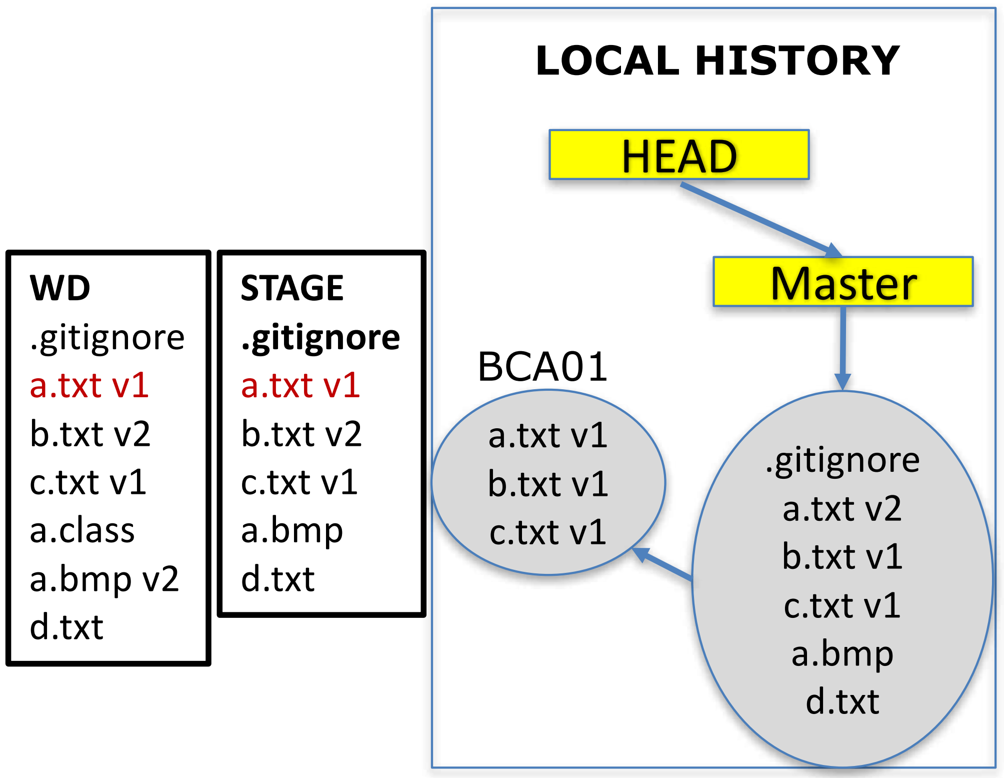
RETRIEVE

git checkout VERSION1

BEFORE checkout



AFTER checkout



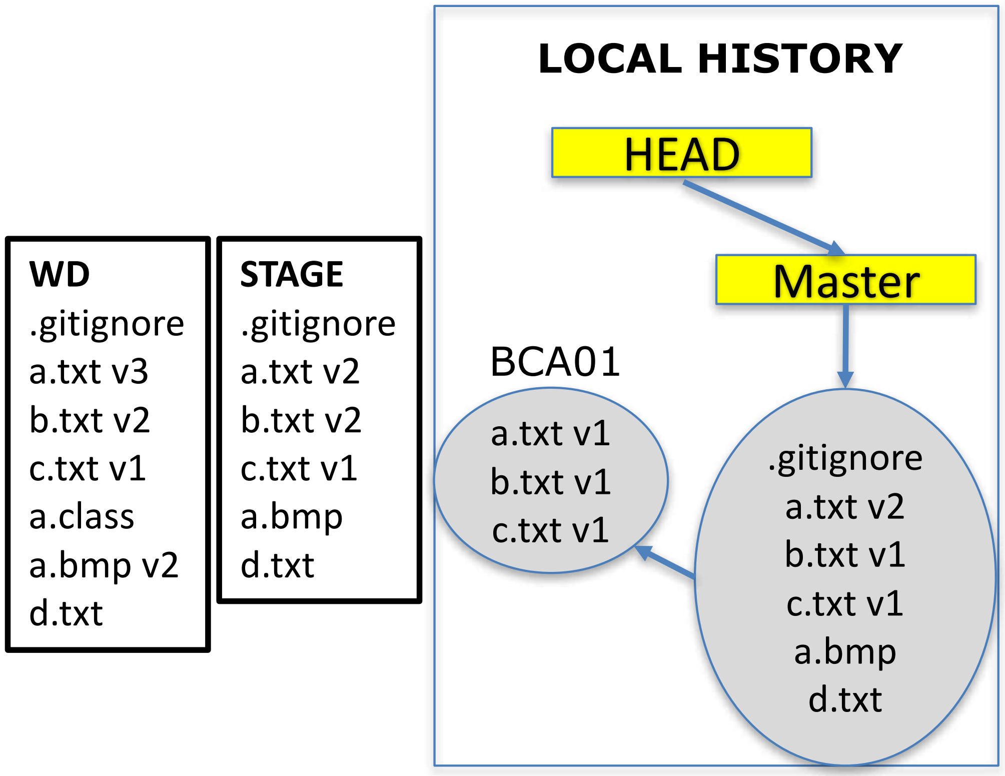
git checkout HEAD~1 a.txt

In this case,
1. ONLY a.txt v1 is retrieved from HEAD~1 to WD and STAGE.
2. Also, HEAD is not changed.

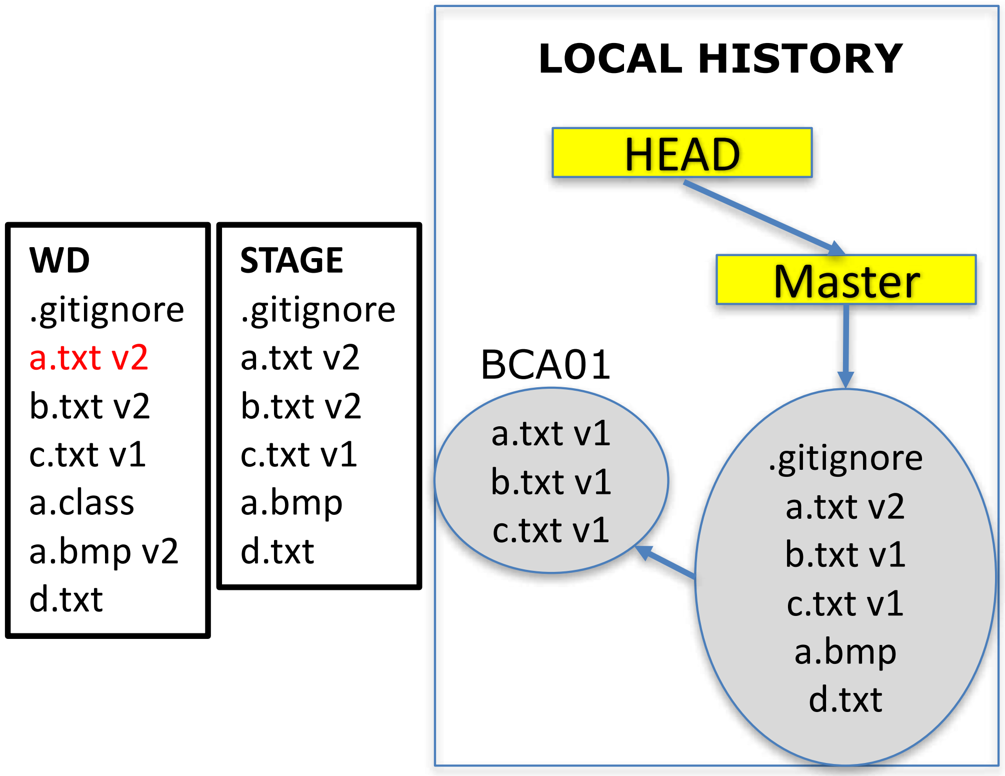
RETRIEVE

git checkout VERSION2

BEFORE checkout



AFTER checkout



git checkout -- a.txt

This version retrieves from STAGE and overwrites WD
(throws away changes to file in WD)
Note use of -- for disambiguation if needed.
In this case, a.txt v2 is written to WD

undo

RESET

UNDO

reset

This command has two versions.

- **Version 1:** git reset COMMIT_OR_BRANCHNAME

This version moves current branch & HEAD pointers to the specified commit. It will also update STAGE to reflect the contents of the new commit.

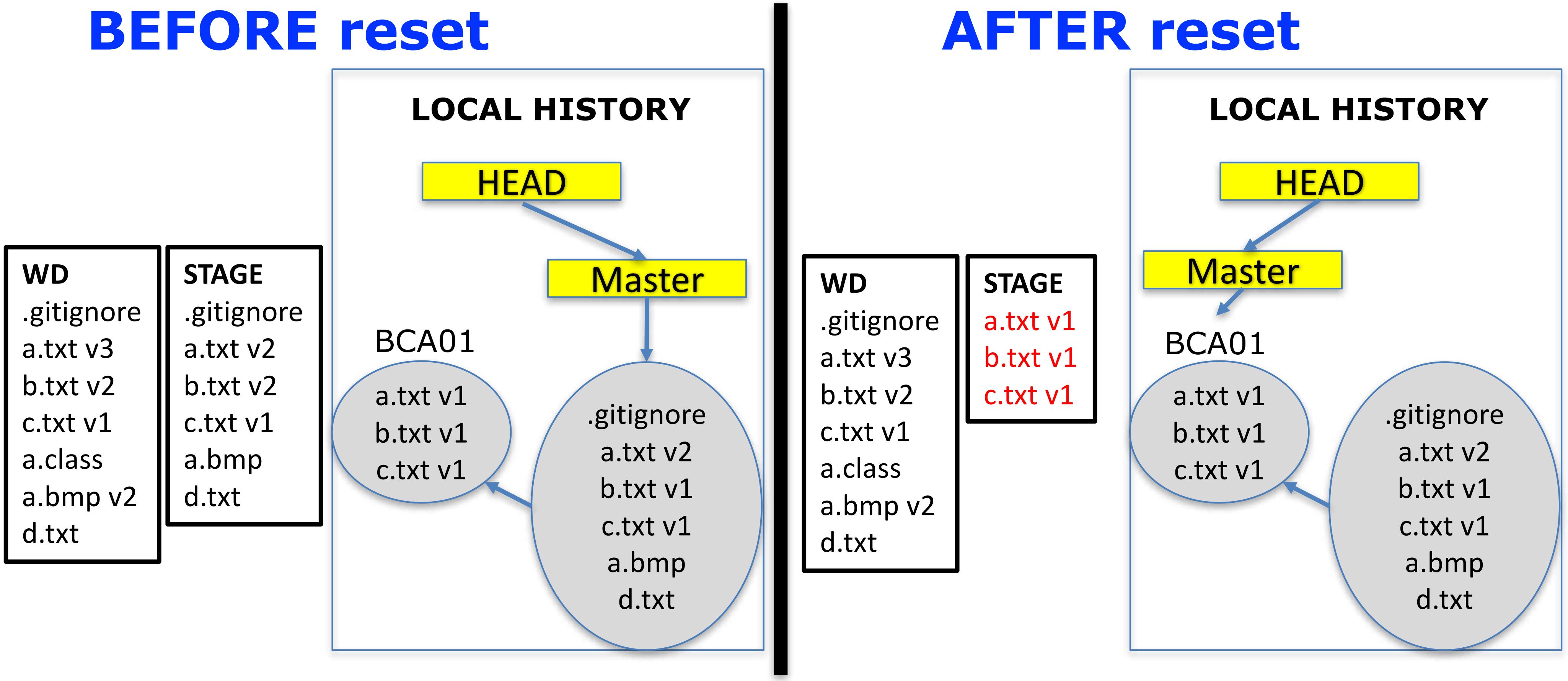
- **Version 2:** git reset FILENAME   (can use -- for disambiguation when needed)

This version retrieves the file(s) from HEAD and overwrites STAGE (throws away changes to file in STAGE).

Also, git reset COMMIT FILENAME will retrieve the file(s) from the COMMIT and overwrite STAGE.

UNDO

git reset VERSION1

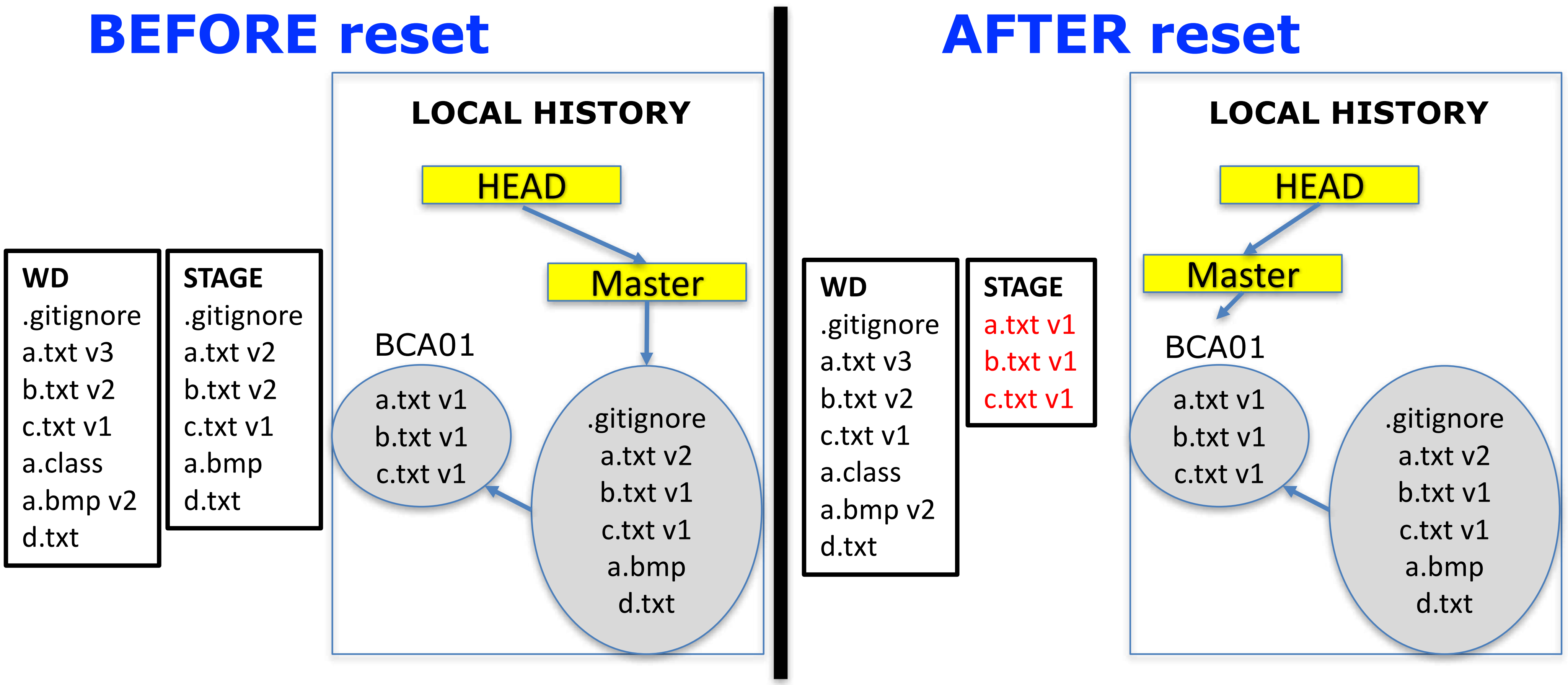


git reset BCA01

1. current branch is changed to BCA01 (like undoing a commit)
2. stage is modified to BCA01
3. WD is untouched.
4. This is like undoing a COMMIT

UNDO

git reset VERSION1



git reset HEAD~1

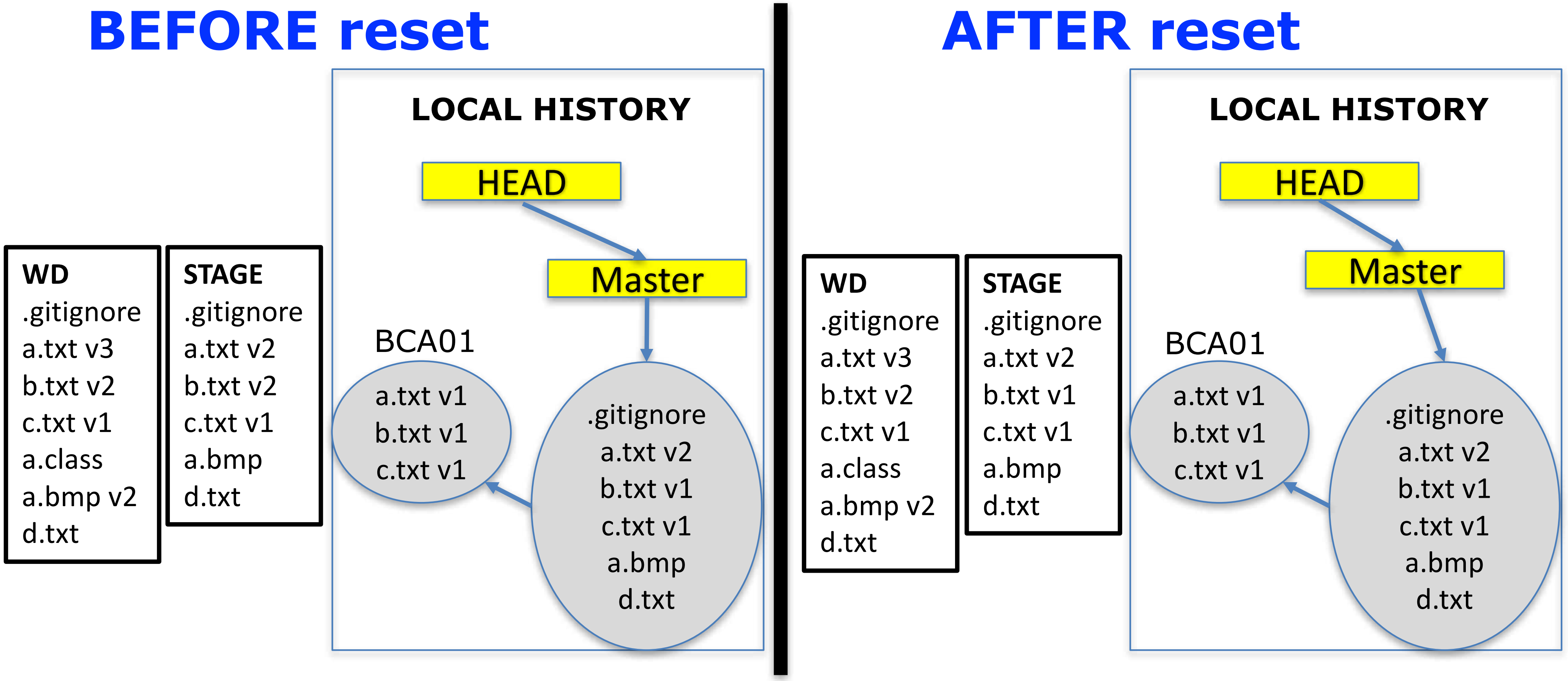
1. HEAD~1 means BCA01
2. current branch is changed to BCA01 (like undoing a commit)
3. stage is modified to BCA01
4. WD is untouched.
5. This is like undoing a COMMIT

UNDO

git reset VERSION1

BEFORE reset

AFTER reset

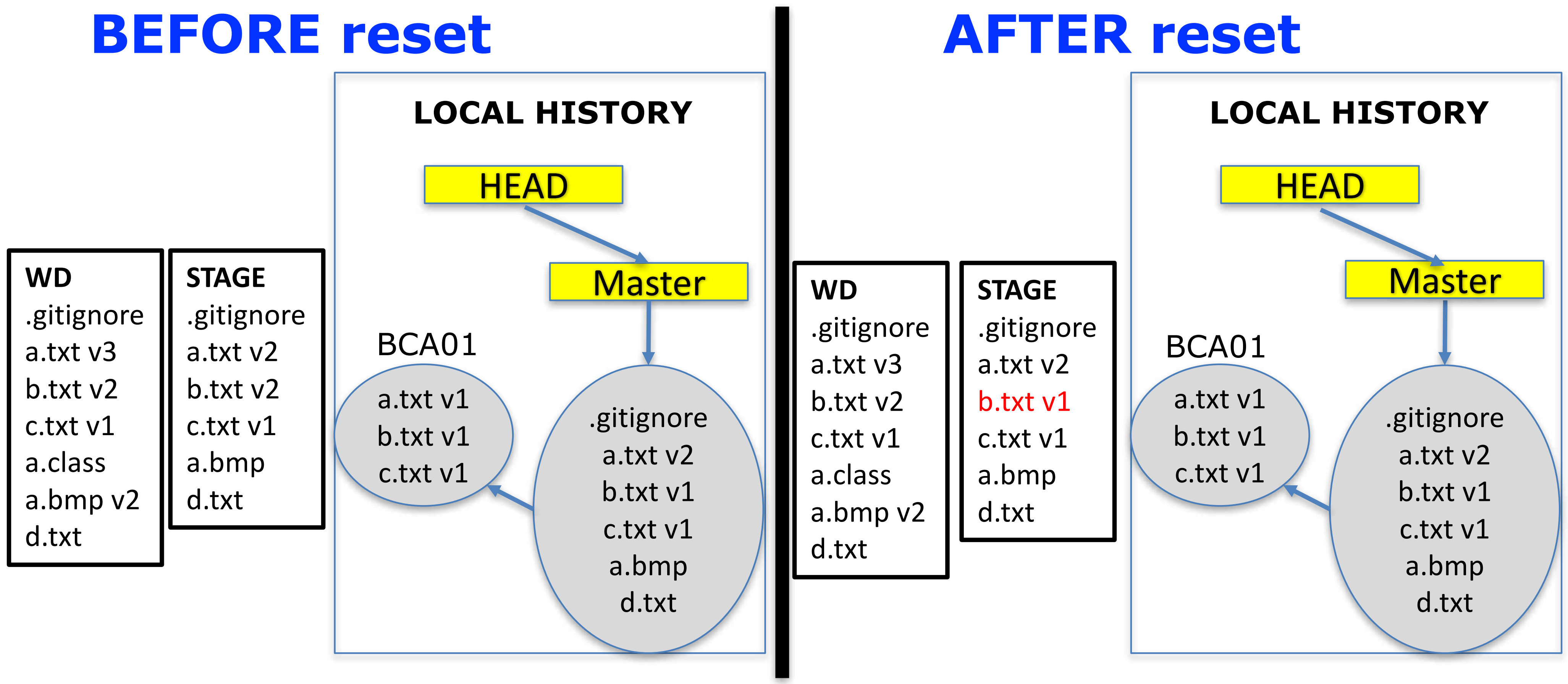


git reset HEAD

1. stage is modified to HEAD
2. WD is untouched.
3. This is like undoing an ADD (i.e. restoring the stage)

UNDO

git reset VERSION2

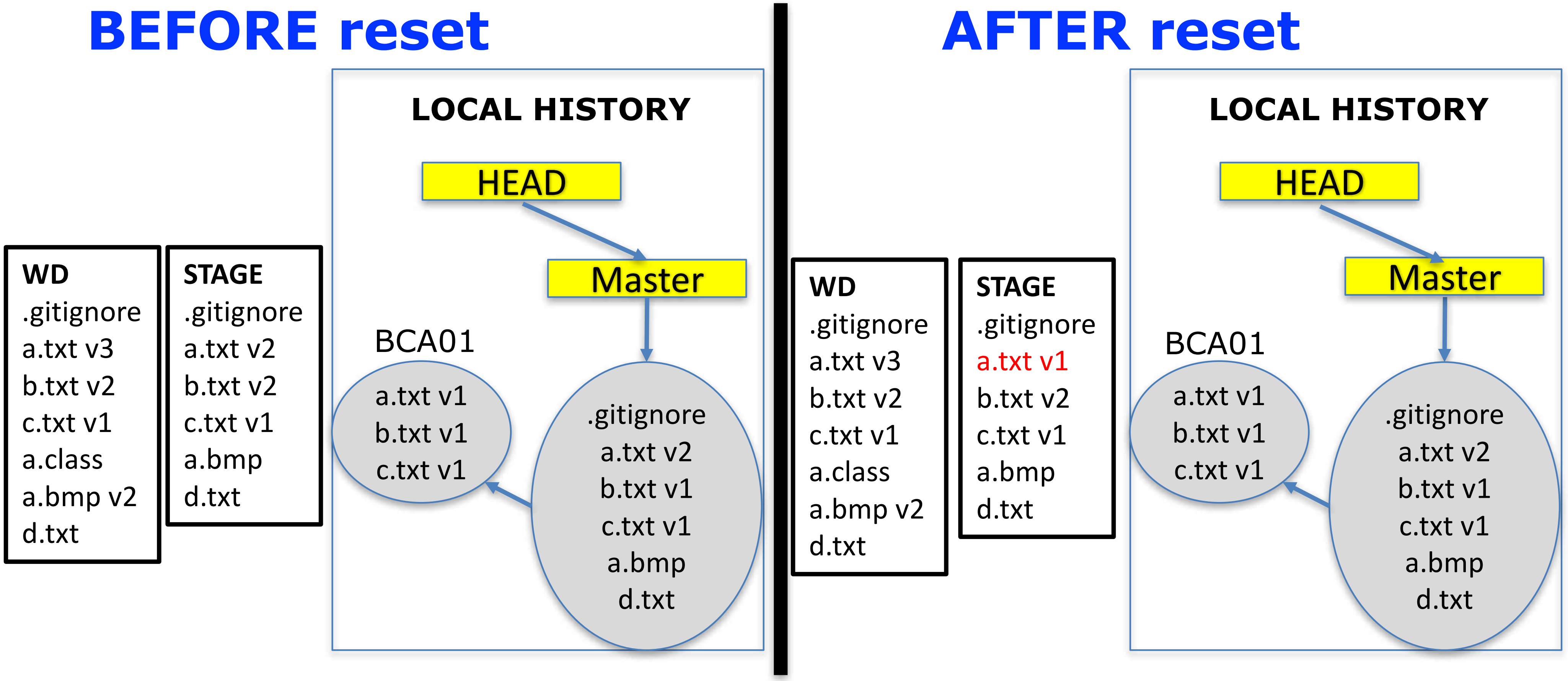


git reset -- b.txt

1. note that files are copied from HEAD in this case.
2. stage is overwritten (Thus, it now has b.txt v1)
3. WD is unchanged
4. This is like undoing an ADD

UNDO

git reset VERSION2

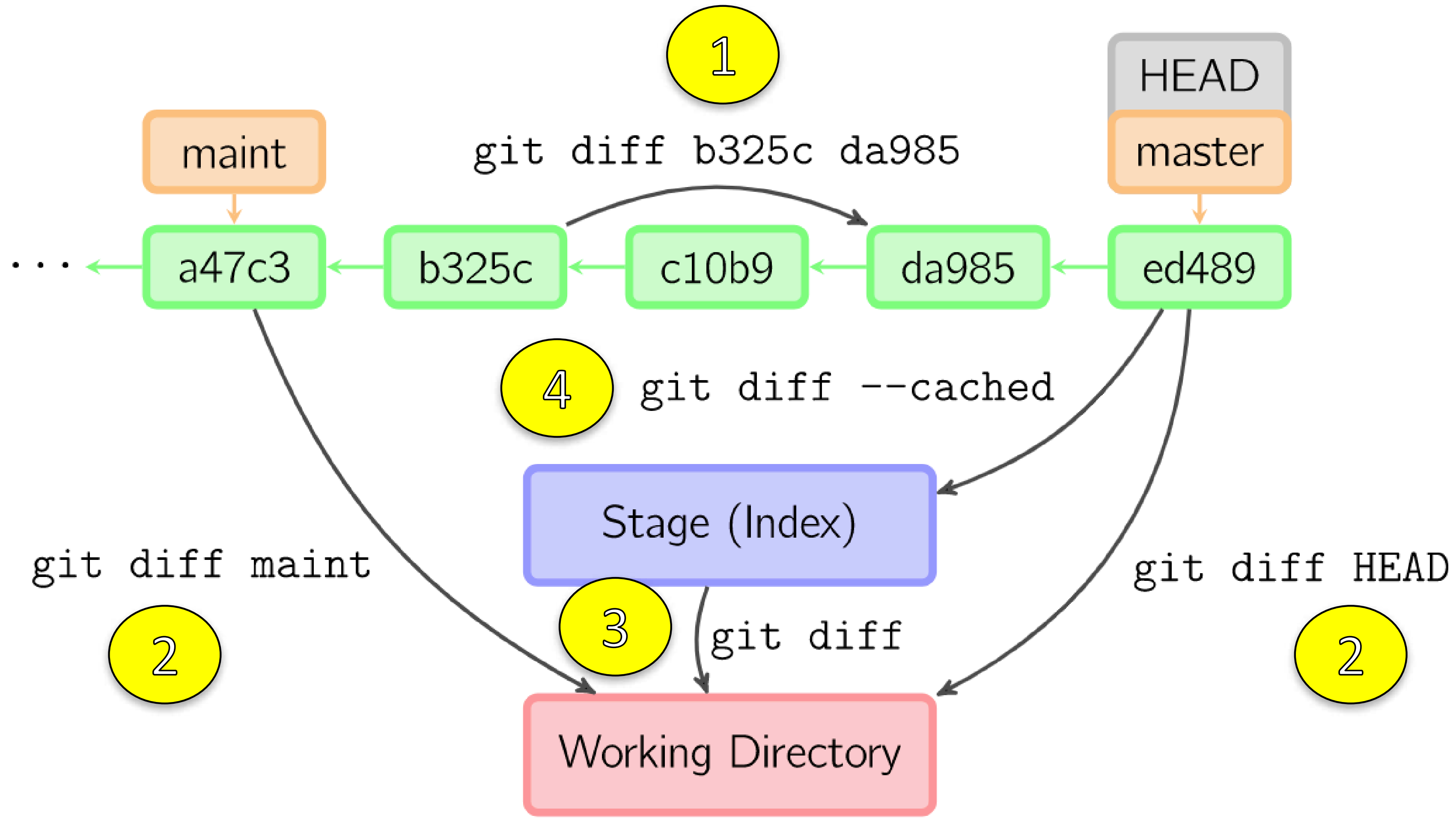


git reset HEAD~1 a.txt

1. note that files are copied from HEAD~1 in this case.
2. stage is overwritten. This it now has a.txt v1
3. WD is unchanged

difference between versions

DIFF



1. diff between two commits (git diff b325c da985)
2. diff between WD and a branch (git diff HEAD)
3. diff between stage and WD (git diff)
4. diff between stage and HEAD (git diff --cached)

THE END!