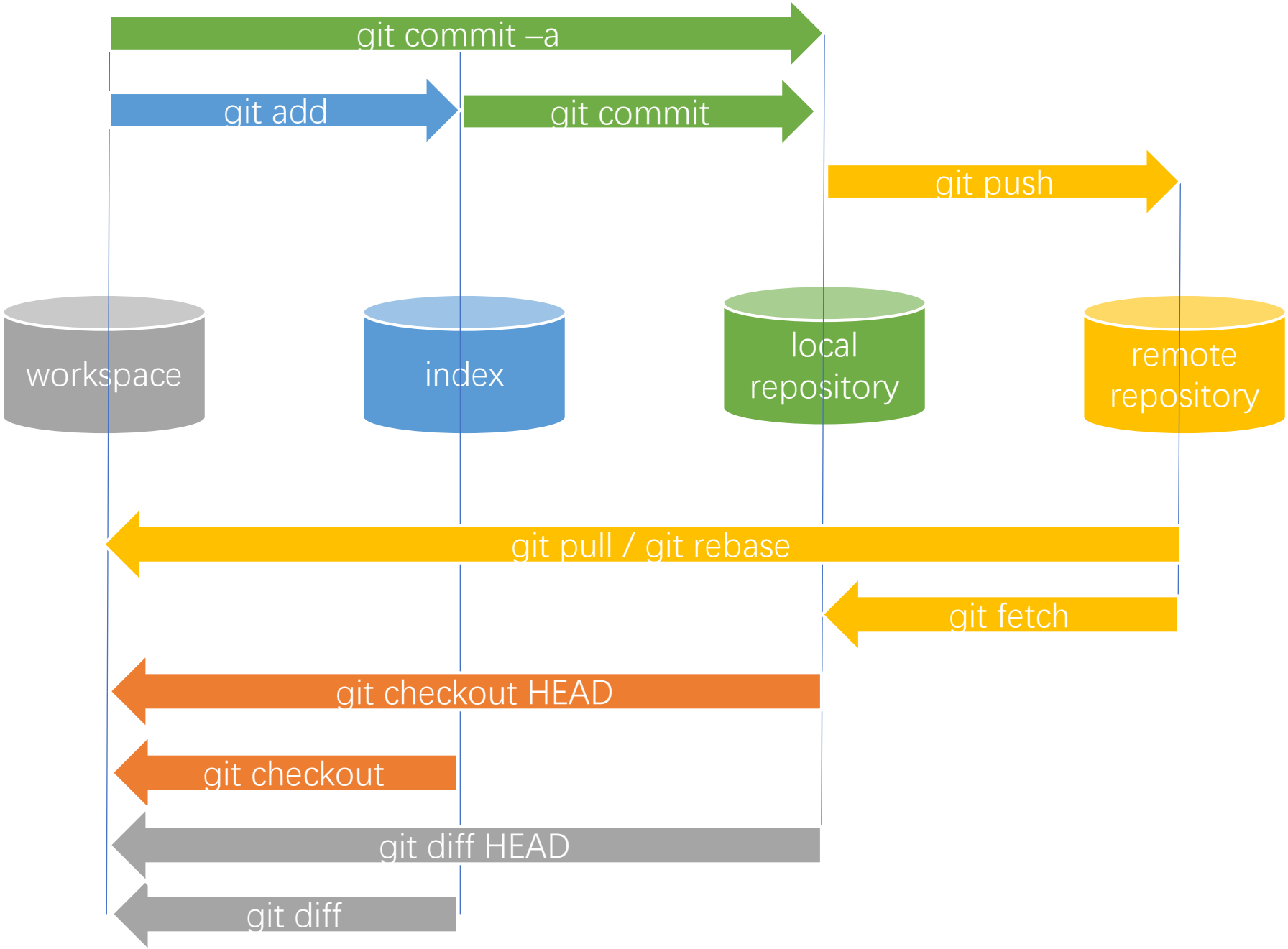
Colorful Git


git commit –a
git add
git commit
git push
workspace
index
local repository
remote repository
git pull / git rebase
git fetch
git checkout HEAD
git checkout
git diff HEAD
git diff

basic: init

- git init
- git fetch remote-repository-url
- git add file
- git commit –m ‘message’
- git push origin master

basic: modify

- modify file
- git add file
- git commit –m ‘modify xxx’

- [warn]: pull before push

- git pull
- git push origin master

basic: view changes

- modify file
- I need to see what changed
- git status

- git add file
- I need to see what changed
- git status

- git commit –m ‘modify xxx’
- Oh shit, I need to see what changed, but `git status` show nothing?
- git status --cached

basic: multi commits

- modify file
- git add file
- git commit –m ‘modify xxx’

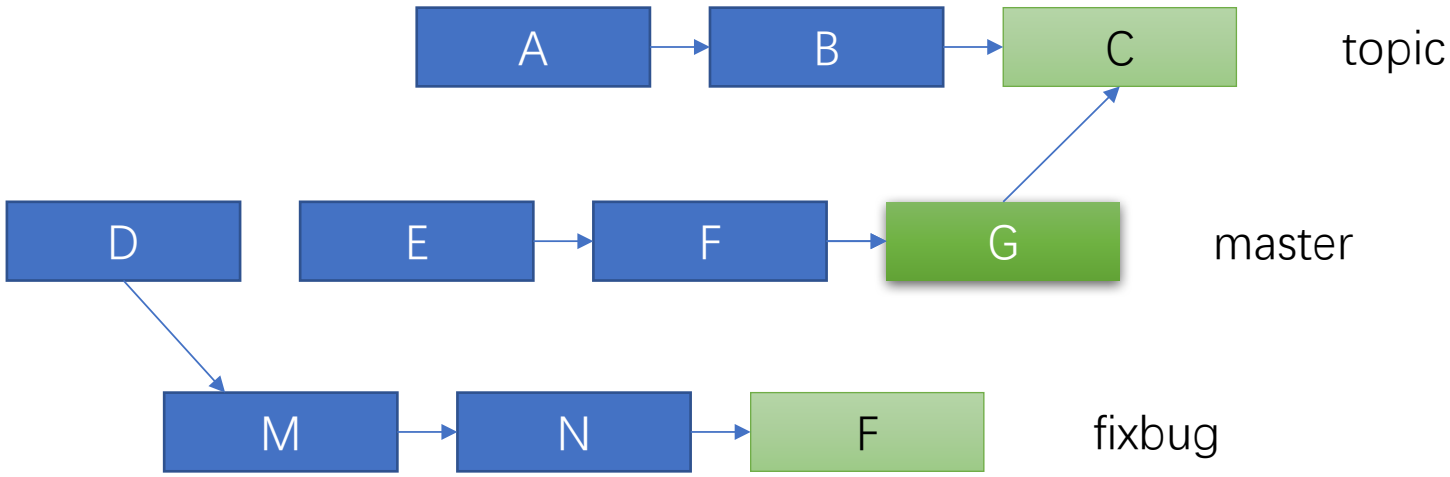
- Oh shit, I committed and immediately realized I need to make one small change!

- git add anotherfile
- git commit ––amend ‘modiy yyy’

branch

- list all branches:
  - git branch
- switch to other branch:
  - git checkout branchname
- create and switch to a new branch:
  - git checkout –b feature_xxx
- delete a branch:
  - git branch –d feature_xxx
- push branch to remote repository
  - git push origin feature_xxx

HEAD



C, G, F are heads, BUT G (which is the current branch head) is HEAD

`F is HEAD~  E is HEAD~~ or HEAD~2, etc`

HEAD

- A head is simply a reference to a commit object. Each head has a name (branch name or tag name, etc). By default, there is a head in every repository called master. A repository can contain any number of heads.

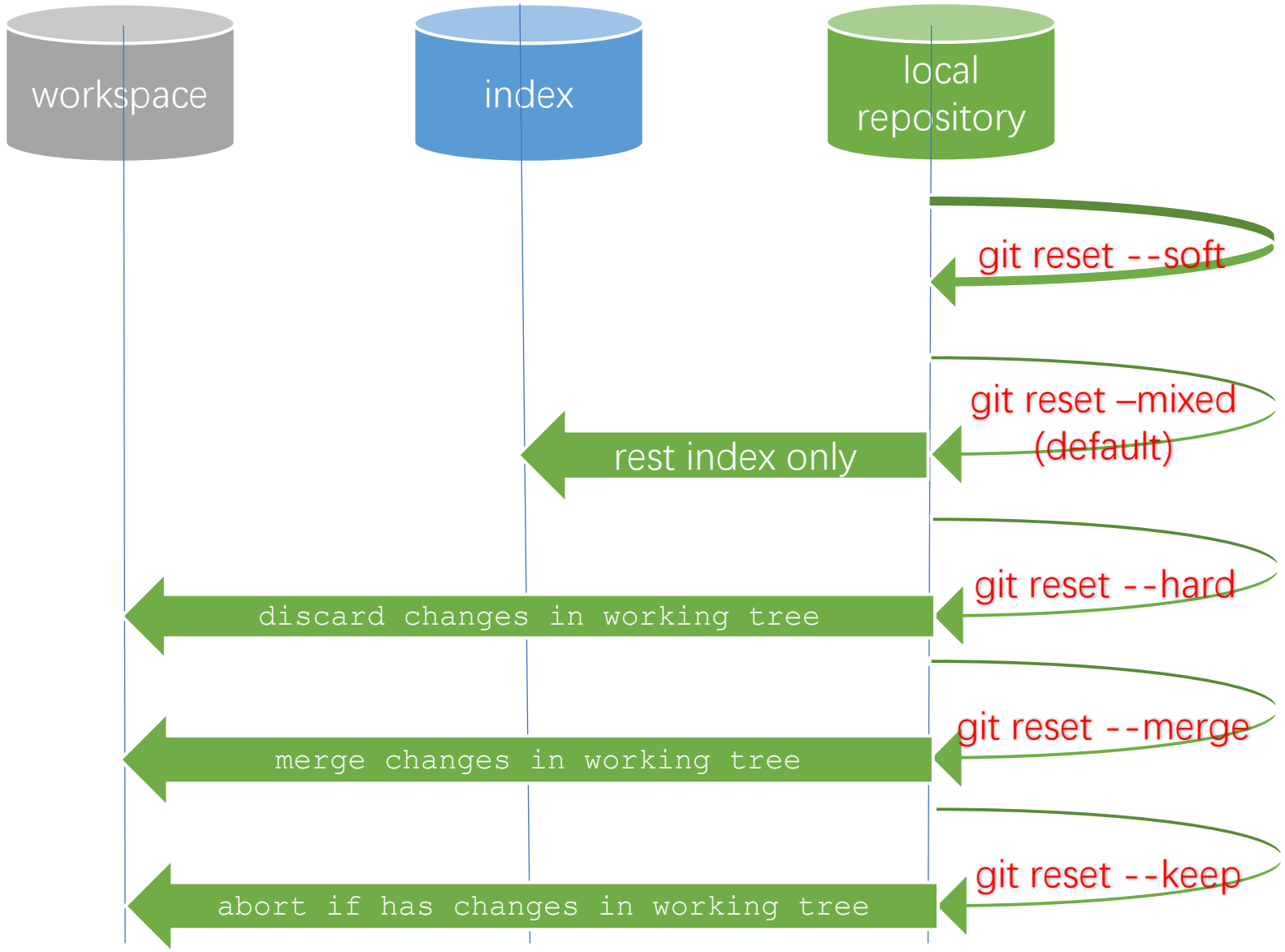
- At any given time, one head is selected as the "current head." This head is aliased to HEAD, always in capitals.

- Note this difference: a "head" (lowercase) refers to any of the named heads in the repository; "HEAD" (uppercase) refers exclusively to the currently active head. This distinction is used frequently in Git documentation.

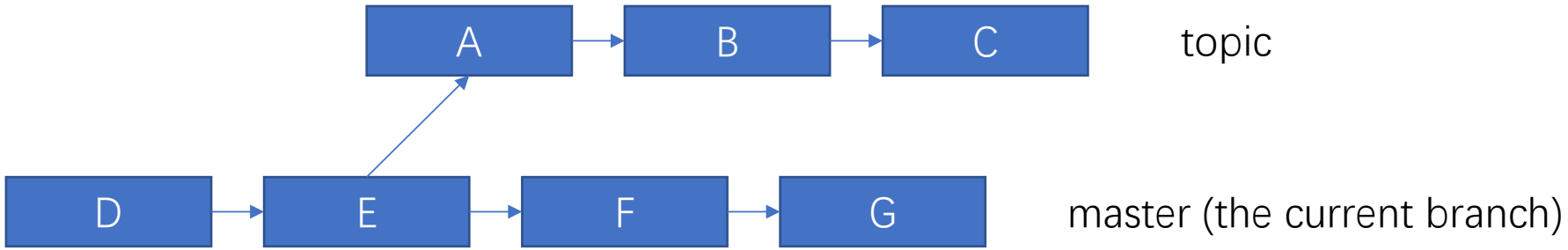
- HEAD is the name given to the commit from which the working tree's current state initialized. In more practical terms, it can be thought of as a symbolic reference to the checked-out commit.

- When HEAD points to a commit that is not the last commit in a branch, that is a "detached HEAD"

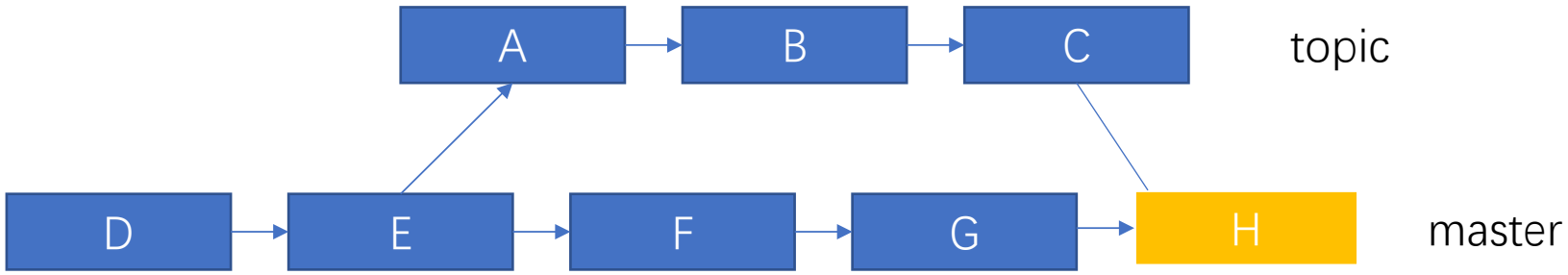
reset


workspace
index
local repository
git reset --soft
git reset –mixed (default)
rest index only
git reset --hard
discard changes in working tree
git reset --merge
merge changes in working tree
git reset --keep
abort if has changes in working tree

merge(1)



`git merge topic` will replay the changes made on the topic branch since it diverged from master(i.e E) until its current commit (C) on top of master, and record the result in a new commit along with the names of the parent commits and a log message from the user describing the changes.



merge(2) resolve conflict

- If conflict when merge, you can
- 1. `git merge --abort`, quit the merge
- 2. resolve the conflict by
  - A. `git merge -s ours` : use the local version
  - B. `git merge –s theirs`: use remote version
  - C. use editor to resolve the conflict directly, see the next page
  - D. use advanced GUI merge tools to resolve conflict
  - last, `git merge --continue` to complete merge.

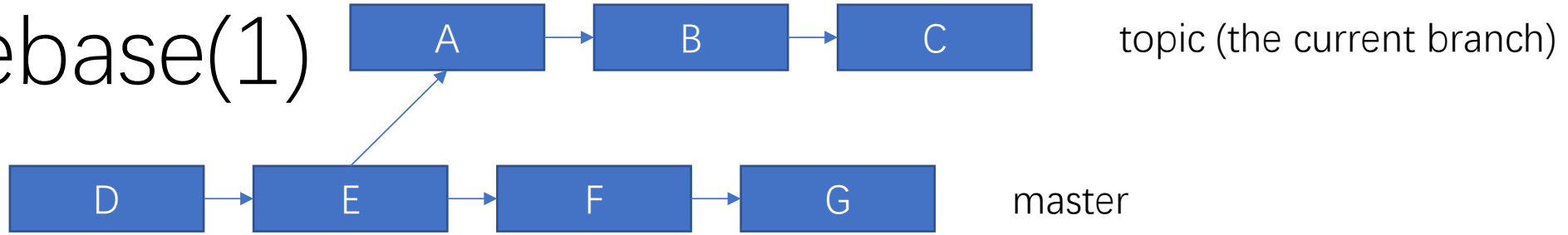
merge(2) resolve conflict by edit directly

```
Here are lines that are either unchanged from the common
ancestor, or cleanly resolved because only one side changed.
<<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
|||||||
Conflict resolution is hard.
=======
Git makes conflict resolution easy.
>>>>>>> theirs:sample.txt
And here is another line that is cleanly resolved or unmodified.
```

ours code

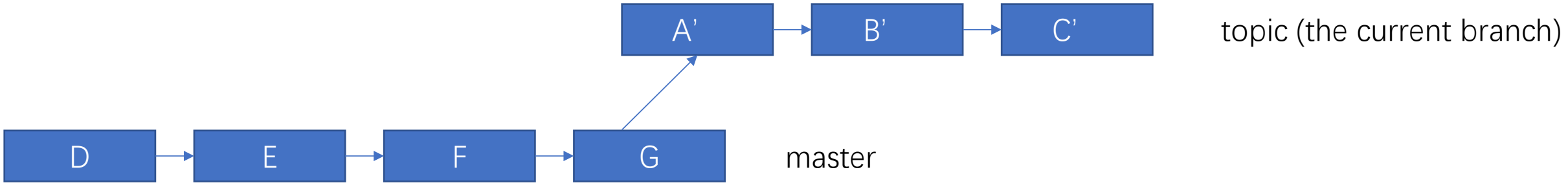
theirs code

rebase(1)



`git rebase <upstream> <branch>` [for example: `git rebase master topic`]

0. if <branch> is set, checkout the <branch> as current branch
1. all changes made by commits in the current branch but that are not in <upstream>
   are saved to a temporary area.
2. the current branch is reset to <upstream>
3. the commits that were previously saved into the temporary area are then reapplied to the
   current branch, one by one, in order.



rebase(2) resolve conflict

the `ours` and `theirs` code are inverse when rebase and merge. WHY?

- If conflict when rebase, you can
- 1. `git rebase --abort`, quit the merge
- 2. resolve the conflict by
  - A. `git rebase -s ours` : use the local version
  - B. `git rebase –s theirs`: use remote version
  - C. use editor to resolve the conflict directly, see the next page
  - D. use advanced GUI merge tools to resolve conflict
  - last, `git rebase --continue` to complete merge.

Rebase(3) resolve conflict

the `ours` and `theirs` code are inverse when rebase and merge. WHY?

```
Here are lines that are either unchanged from the common
ancestor, or cleanly resolved because only one side changed.
<<<<<<< yours:sample.txt
Conflict resolution is hard;
let's go shopping.
|||||||
Conflict resolution is hard.
=======
Git makes conflict resolution easy.
>>>>>>> theirs:sample.txt
And here is another line that is cleanly resolved or unmodified.
```

ours code

theirs code

pull

- git pull = git fetch + git merge
- git pull –rebase = git fetch + git rebase

conflict when merge

- git add another-file
  - or, add all changes files by git add .
- git merge / git pull
- something conflict, then:
  - the current branch is a anonymous branch, use `git status` to see
  - you can name the anonymous branch, use `git checkout –b name` to save it.
- => resolve the conflict by merge tool
  - git add .
  - git commit -m ‘resolve conflict’
  - git merge --continue
- => or discard the merge process
  - git merge --abort

conflict when rebase

- git add another-file
  - or, add all changes files by git add .
- git rebase/ git pull --rebase
- something conflict, then:
  - the current branch is a anonymous branch, use `git status` to see
  - you can name the anonymous branch, use `git checkout –b name` to save it.
- => resolve the conflict by merge tool
  - git add .
  - git commit -m ‘resolve conflict’
  - git rebase --continue
- => or discard the pull process
  - git rebase --abort

cherry-pick(1)

- Apply the changes introduced by some existing commits.
- Examples:
- **git cherry-pick master**
  - Apply the change introduced by the commit at the tip of the master branch and create a new commit with this change.
- **git cherry-pick master~4 master~2**
  - Apply the changes introduced by the fifth and third last commits pointed to by master and create 2 new commits with these changes.
- [more…]: https://git-scm.com/docs/git-cherry-pick

cherry-pick(2) resolve conflict

- resolve conflict as merge
- git cherry-pick --continue
- git cherry-pick --abort

log basic

- git log
- git log --author=ffl
- git log --pretty=oneline
- git log --graph --online --decorate --all
- git log --help

git lm / git lms /git ls /git lss

- git config --global alias.lm "log --no-merges --color --date=format:'%Y-%m-%d %H:%M:%S' --author='fanfeilong' --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Cblue %s %Cgreen(%cd) %C(bold blue)<%an>%Creset' --abbrev-commit”
- git config --global alias.lms "log --no-merges --color --stat --date=format:'%Y-%m-%d %H:%M:%S' --author='fanfeilong' --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Cblue %s %Cgreen(%cd) %C(bold blue)<%an>%Creset' --abbrev-commit"

- git config --global alias.ls "log --no-merges --color --graph --date=format:'%Y-%m-%d %H:%M:%S' --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Cblue %s %Cgreen(%cd) %C(bold blue)<%an>%Creset' --abbrev-commit"
- git config --global alias.lss "log --no-merges --color --stat --graph --date=format:'%Y-%m-%d %H:%M:%S' --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Cblue %s %Cgreen(%cd) %C(bold blue)<%an>%Creset' --abbrev-commit"

END