

GIT BRANCHING

Architecture & Deployment

Speaker notes

Learn how to work on isolated, parallel lines of development with [Git](#) branches.

This is a condensed version of the [branching chapter of the Git Book](#), which you should read if you want more detailed information on the subject.

You will need

- A Unix CLI
- [Git](#)

Recommended reading

- [Version control with Git](#)

Resources

- [Git branching](#)
- [Advanced merging](#)
- [Understanding branches in Git](#)
- [Branching workflows](#)
 - [A successful branching model](#) (for large teams)
 - [A successful branching model considered harmful](#)
 - [Branch-per-feature](#)
 - [Trunk-based development](#)

WHAT IS BRANCHING?



Branching means you **diverge from the main line of development** and continue to do work without messing with that main line.

WHY USE BRANCHES?

- Work **in isolation**
- Pull changes from the main line **at your own pace**
- Choose **which features to release and when**

Speaker notes

Git has a very powerful branching model that is very **lightweight and fast**: it encourages workflows that branch and merge often.

Many teams using Git create a **separate branch** to develop **each feature**.

REMEMBER COMMITS?



387f12

T c0252f



9ab3fd

T b458d4



4f94fa

T a15551

Speaker notes

Remember that Git stores data as a series of snapshots.

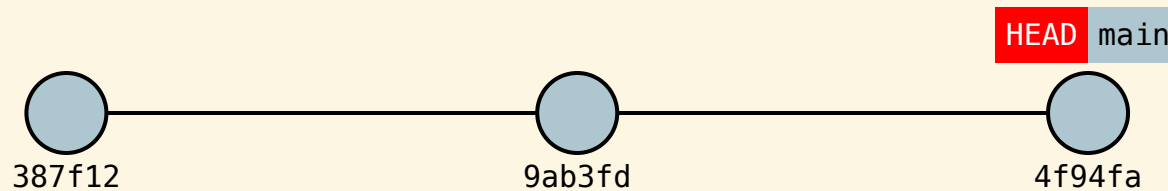
Each **commit** (the circles above) contains a pointer to the snapshot of the content you staged, represented by the blue **T**ree rectangles (as they refer to a *tree* of file snapshots).

Each commit also contains:

- The user name and e-mail or the author
- The date at which the commit was created
- A pointer to the previous commit (or commits)

BRANCHES POINT TO COMMITS

A branch is a lightweight, movable **pointer to a commit**.



Speaker notes

The default branch is `main` (or `master` with the default Git configuration). The special `HEAD` pointer indicates the current branch.

As you start making commits, the current branch pointer **automatically moves** forward to your latest commit.

EXAMPLE REPOSITORY

We will use a prepared repository to illustrate branching. Clone it and check it out.

```
$> cd /path/to/projects  
  
$> git clone https://github.com/ArchidDep/git-branching-ex.git  
  
$> cd git-branching-ex  
  
# We will talk more about this  
$> git remote rm origin
```

Speaker notes

As you can see if you type `git log`, there are some commits already. Open the project with your favorite editor and open the `index.html` page in a browser.

WORKING WITH BRANCHES

SHOWING BRANCHES ON THE COMMAND LINE

```
$> git log --oneline --decorate --graph --all  
* 4f94fa (HEAD -> main) Improve layout  
* 9ab3fd Fix addition  
* 387f12 First version
```

Speaker notes

The `git log` command can show you a representation of the commit graph and its branches.

CREATING GIT ALIASES

```
$> git config --global alias.graph \  
    "log --oneline --decorate --graph --all"  
  
$> git graph  
* 4f94fa (HEAD -> main) Improve layout  
* 9ab3fd Fix addition  
* 387f12 First version
```

Speaker notes

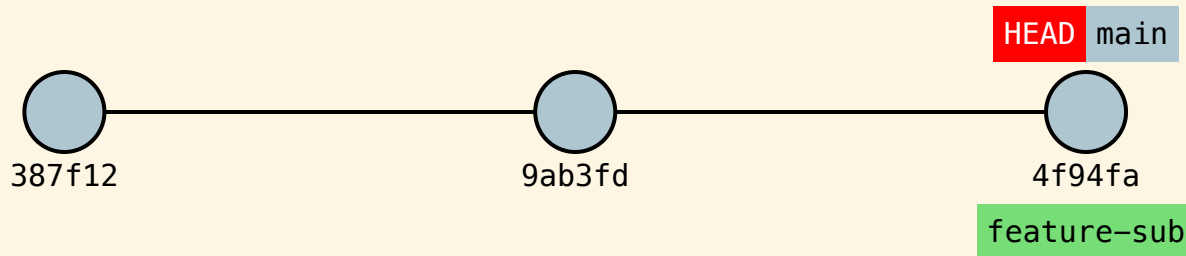
In fact, this command is so useful you should make an **alias**, as we will use it a lot in this tutorial.

CREATE A NEW BRANCH



*Our JavaScript calculator is missing some code.
Let's create a branch to implement subtraction.*

```
$> git branch feature-sub
```



Speaker notes

It's very fast and simple to create a new branch. Use the `git branch` command to create a branch called "feature-sub":

There is now a new pointer to the current commit. Note that `HEAD` didn't move – we are still on the `main` branch.

SHOWING THE CURRENT BRANCH

```
$> git branch  
* main  
feature-sub
```

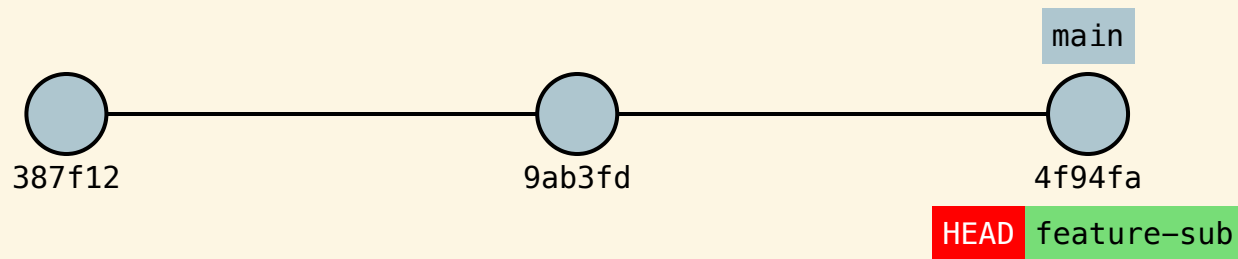
Speaker notes

You can use `git branch` without arguments to simply see the list of branches and which one you are currently on:

The star is displayed next to the current branch.

SWITCH BRANCHES

```
$> git switch feature-sub # or git checkout feature-sub  
Switched to branch 'feature-sub'
```



 You can now implement the subtraction in `subtraction.js`. Move on to the next slide once you're done.

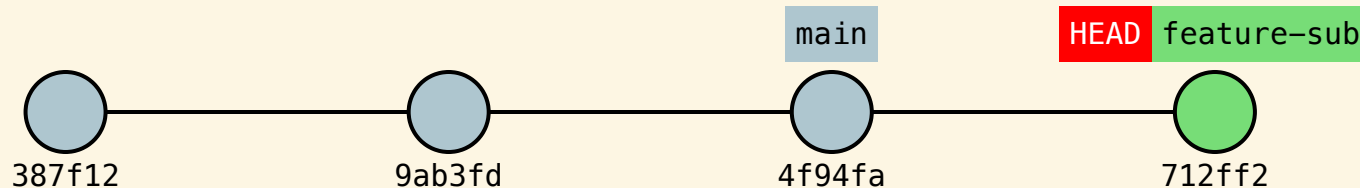
Speaker notes

Now let's switch to the `feature-sub` branch:

This moves `HEAD` to point to the `feature-sub` branch. Nothing else happened because `HEAD` is still pointing to the same commit as `main`.

COMMIT ON A BRANCH

```
$> git add subtraction.js  
  
$> git commit -m "Implement subtraction"  
[feature-sub 712ff2] Implement subtraction  
1 file changed, 1 insertion(+), 1 deletion(-)
```



Speaker notes

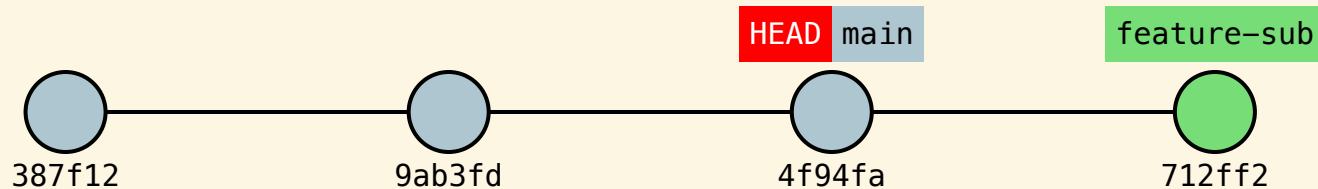
Once you're done, it's time to add and commit your changes. As you commit, the current branch (the one pointed to by HEAD) moves forward to the new commit.

SWITCH BACK TO `main`

 Oops, you just noticed that addition is not working correctly. You need to make a bug fix, but you don't want to mix that code with the new subtraction feature. Let's **go back to** `main`.

SWITCH/CHECKOUT BEHAVIOR

```
$> git switch main # or git checkout main  
Switched to branch 'main'
```



Now check your files.

Speaker notes

Two things happened when you ran `git switch main` (or `git checkout main`):

- The HEAD pointer was **moved** back to the main branch.
- The files in your working directory were **reverted** back to the snapshot that main points to.

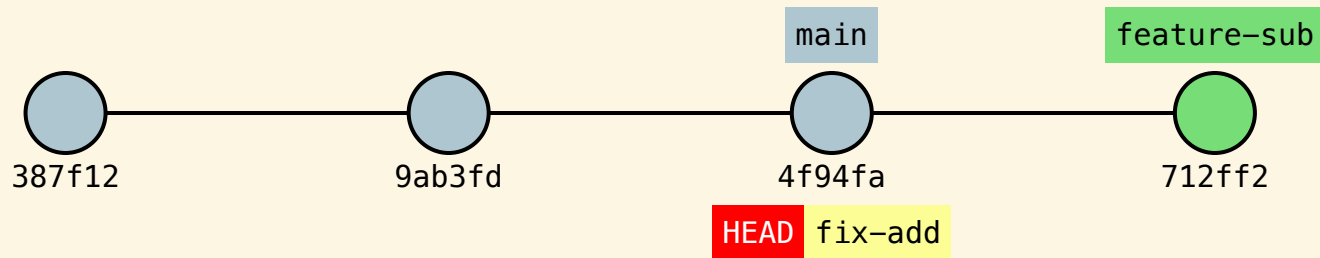
You have essentially **rewinded** the work you've done in `feature-sub`, and are working on an **older version** of the project.

CREATE ANOTHER BRANCH



Let's create a new branch to fix the bug.

```
$> git switch -c fix-add # or git checkout -b fix-add  
Switched to a new branch 'fix-add'
```



Speaker notes

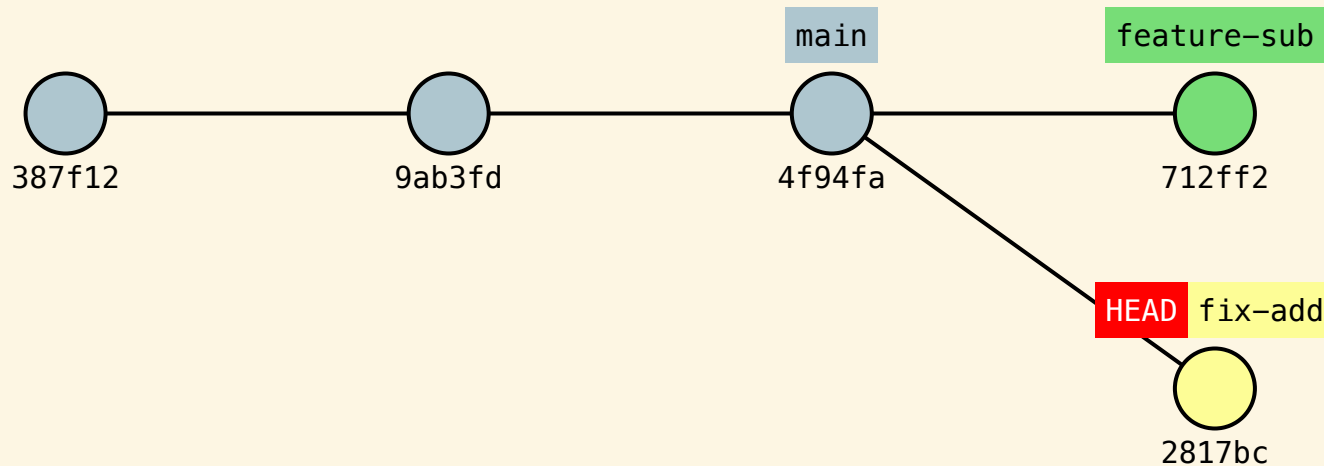
You can create a new branch *and* switch to it in one command with the `-c` (**c**reate) option of the `switch` command or the `-b` (new **b**ranch) option of the `checkout` command.

Nothing changed yet because `fix-add` still points to the same commit as `main`.

WORK ON A SEPARATE BRANCH

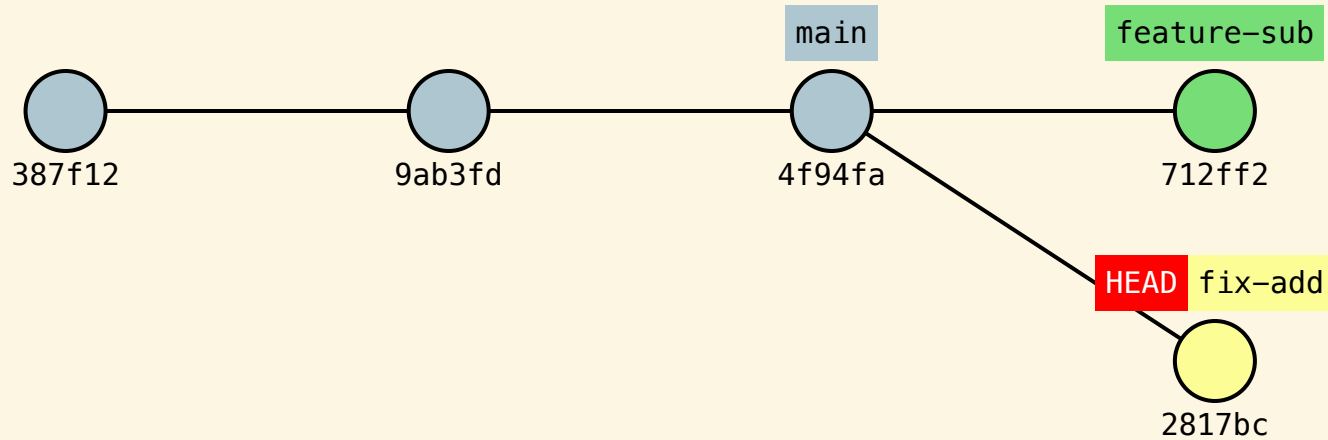
🔧 Fix `addition.js` and commit your changes.

```
$> git add addition.js  
$> git commit -m "Fix addition"  
[fix-add 2817bc] Fix addition  
1 file changed, 1 insertion(+), 1 deletion(-)
```



DIVERGENT HISTORY

```
$> git switch feature-sub  
$> git switch fix-add
```



Speaker notes

Your project history has now **diverged**.

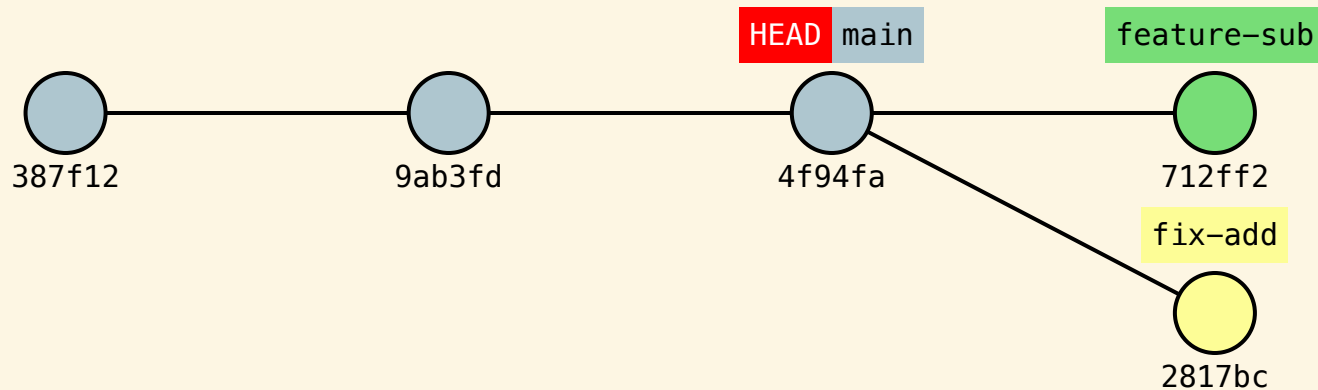
The changes in `feature-sub` and `fix-add` are **isolated**. You can **switch back and forth** between the branches with `git switch` or `git checkout`:

Every time you switch to one of these branches, the files in your **working directory** are updated to reflect the state of the corresponding commit, or snapshot.

MERGING

Let's bring back those changes to the main line.

```
$> git switch main # or git checkout main
```



Speaker notes

Now that you've tested your fix and made sure it works, you want to **bring those changes** back into the **main** branch.

Git's **merge** command can do that for you, but it can only **bring changes** from another branch **into the current branch**, not the other way around. So you must first switch to the **main** branch.

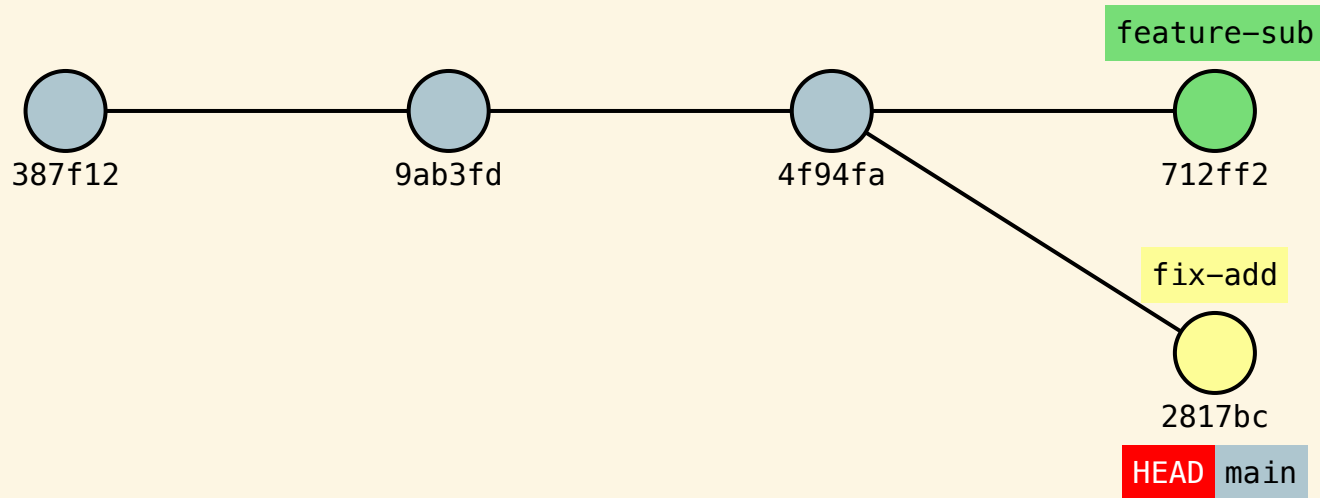
MERGE A BRANCH

Merge the changes from the `fix-add` branch:

```
$> git merge fix-add
Updating 4f94fa..2817bc
Fast-forward
 addition.js | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Notice the term **fast-forward**.

FAST-FORWARD

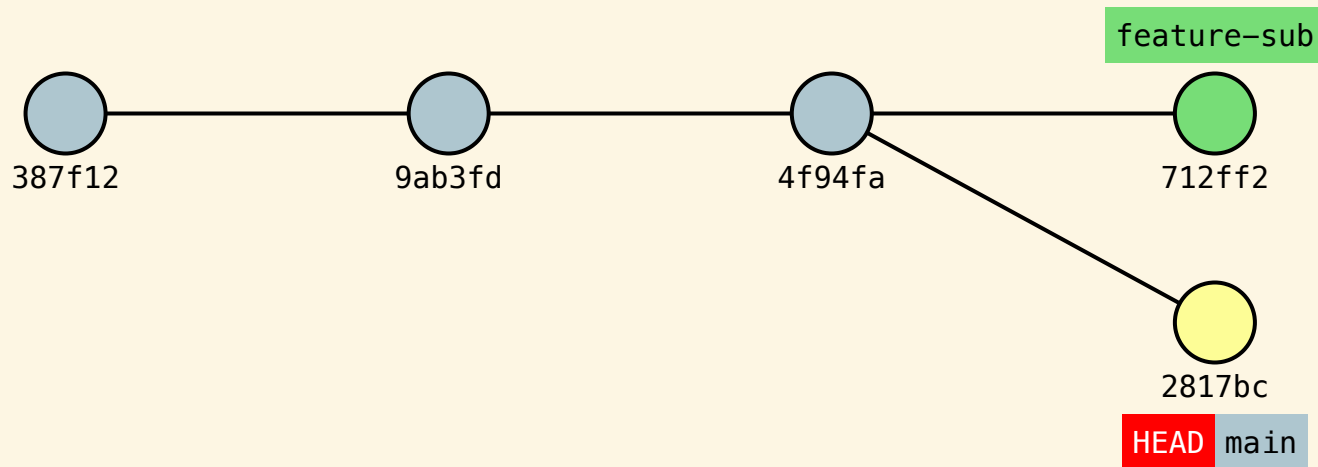


Speaker notes

The `fix-add` branch pointed to a commit **directly ahead** of the commit `main` pointed to. There is no divergent history, so Git simply has to **moves the pointer forward**. This is what is called a **fast-forward**.

DELETE A BRANCH

```
$> git branch -d fix-add  
Deleted branch fix-add (was 2817bc).
```



Speaker notes

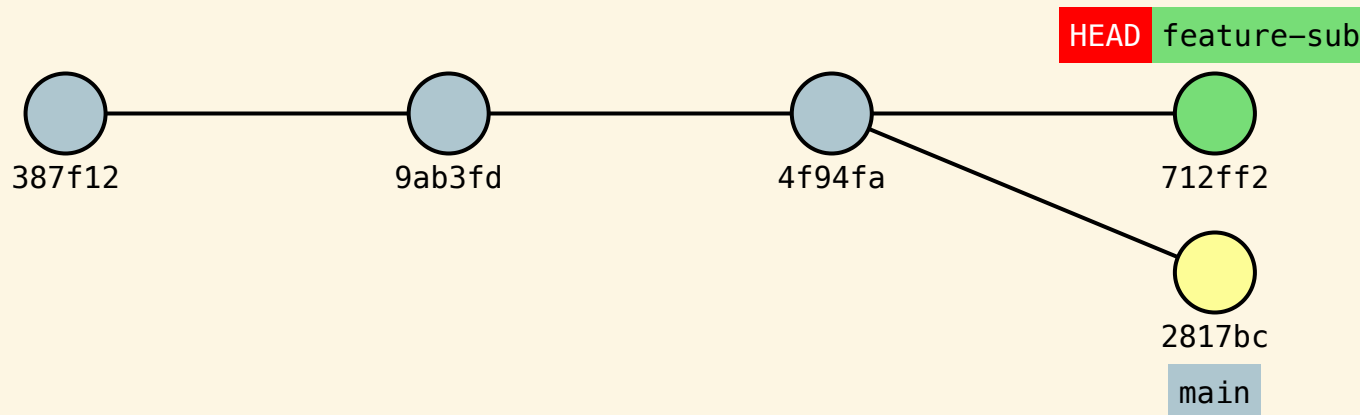
Now that we've brought our fix back into `main`, we don't need the `fix-add` branch anymore. Let's delete it with the `-d` (**d**elete) option of the `branch` command:

CONTINUE WORKING ON A FEATURE BRANCH



Let's switch back to our `feature-sub` branch and finish our work. As good programmers, we need to write a comment for the subtract function.

```
$> git switch feature-sub # or git checkout feature-sub
```

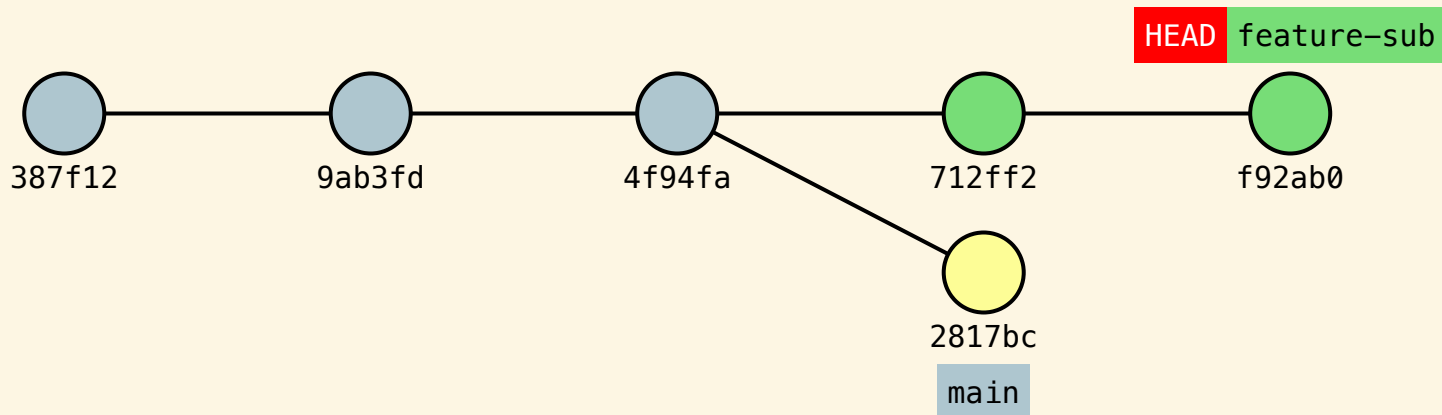


COMMIT YOUR CHANGES

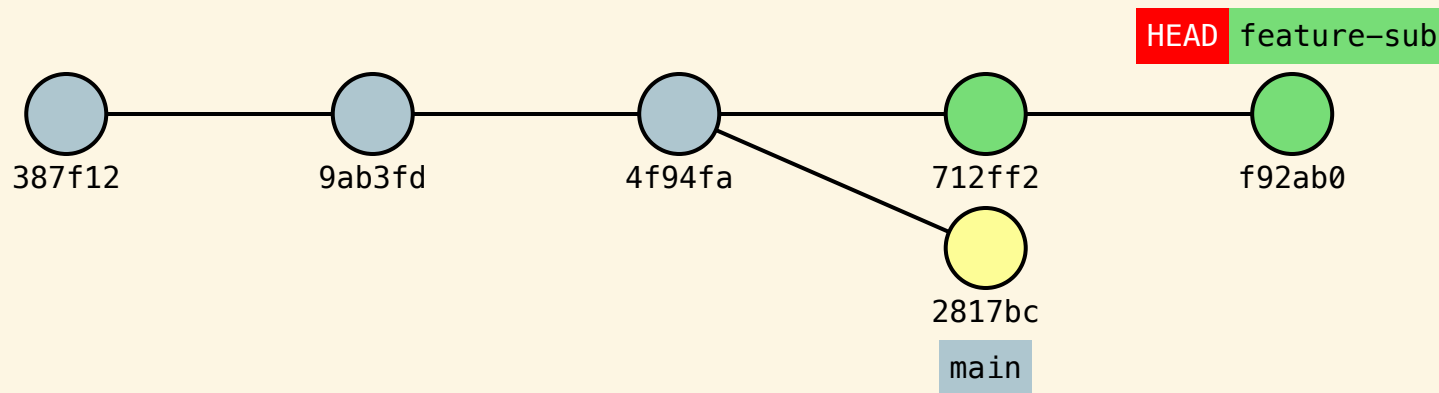


Once you are done, commit your changes.

```
$> git add subtraction.js  
$> git commit -m "Comment subtract function"
```



MERGING A DIVERGENT HISTORY



Oops, no fast-forward here.

Speaker notes

Now that we're happy with our new subtraction feature, we want to **merge** it into `main` as well. But the `feature-sub` branch has **diverged from some older point compared to** `main`, so Git cannot do a fast-forward:

- `feature-sub` points to commit `f92ab0` which contains our feature.
- `main` points to commit `2817bc` which contains the addition fix.
- Commit `4f94fa` is the common ancestor.

Git will do a **three-way merge** instead, combining together the changes of `main` and `feature-sub` (compared to the common ancestor). A **new commit** will be created representing that state.

MERGE THE DIVERGENT BRANCH



Switch back to the `main` branch and merge `feature-sub` into it.

```
$> git switch main # or git checkout main
$> git merge feature-sub
Merge made by the 'recursive' strategy.
 subtraction.js | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
```

MERGE COMMIT MESSAGE

Git will ask you to confirm the commit message:

```
Merge branch 'feature-sub'

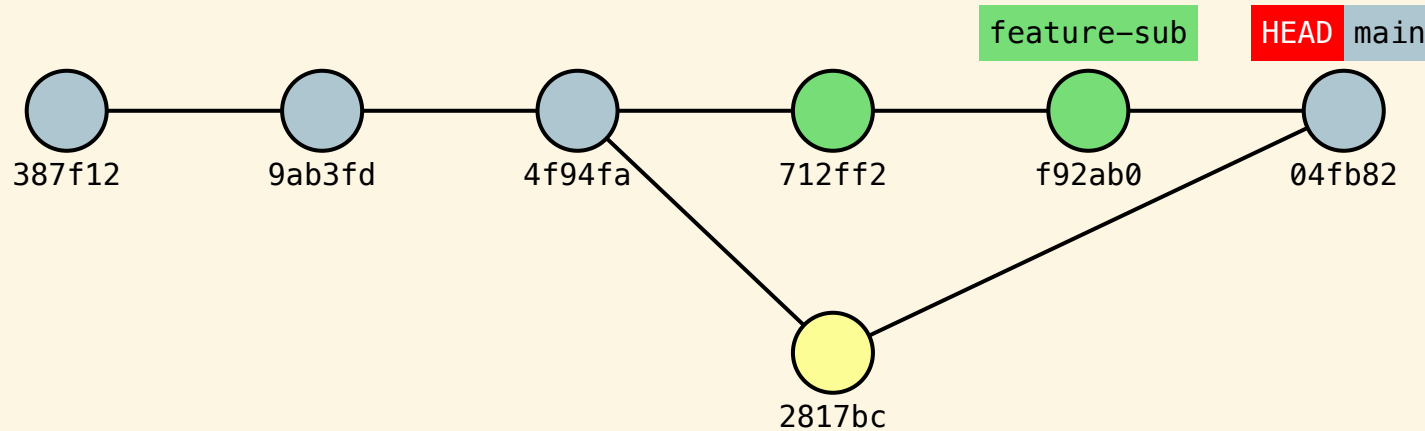
# Please enter a commit message to explain why this merge is
# necessary, especially if it merges an updated upstream into
# a topic branch.
#
# Lines starting with '#' will be ignored, and an empty
# message aborts the commit.
```

If you are in Vim, type `:wq` (**w**rite and **q**uit) to save and exit. If you are in nano, use `Ctrl-X`.

Speaker notes

Git will need to create a new commit when you run the `merge` command, so it will **open the configured editor** (Vim by default if you have not changed it) with a generated commit message.

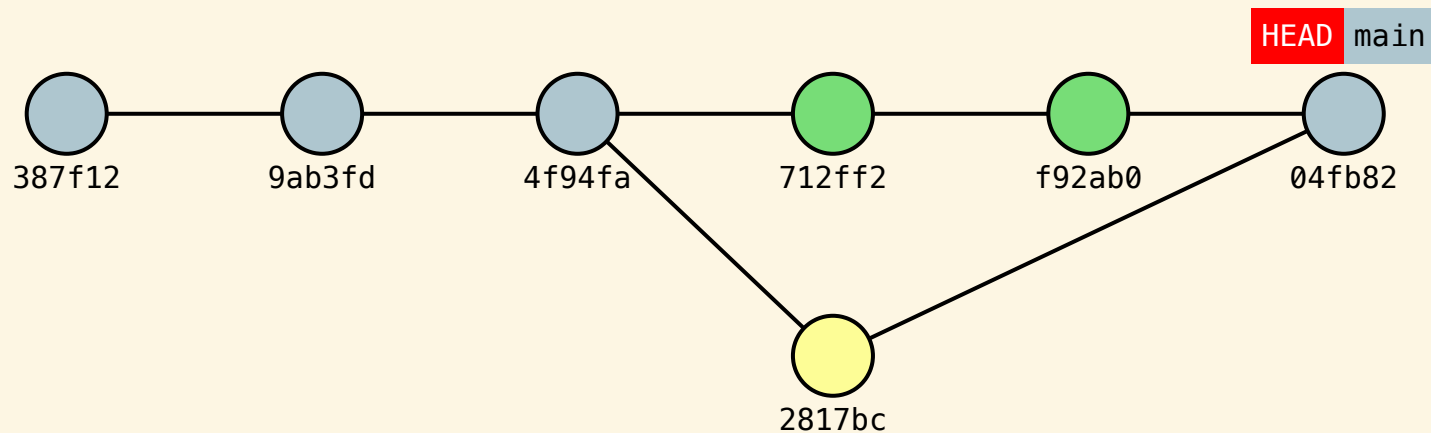
MERGE COMMIT



You can see the new **merge commit** that Git has created. It is a special commit in that it has more than one parent.

DELETE feature-sub

```
$> git branch -d feature-sub
```



MERGE CONFLICTS

Occasionally, the merge process doesn't go smoothly: if the **same line(s) in the same file(s)** was modified in two diverging branches and you merge them together, Git can't know which is the correct version.

CREATE SOME CONFLICT

Let's pretend that a colleague of yours also implemented the subtraction function but in a different way than you did.



It must have been a colleague... you weren't that drunk last night.

FIND THE COMMON ANCESTOR

Let's find our original starting point (the common ancestor where `feature-sub` and `fix-add` diverged) and start a new branch from there.

```
1 $> git graph
2 *    04fb82 (HEAD -> main) Merge branch 'feature-sub'
3   |\
4   | * f92ab0 Comment subtract function
5   * | 2817bc Fix addition
6   | * 712ff2 Implement subtraction
7   |/
8   * 4f94fa (origin/main, origin/HEAD) Comment add function
9   * 9ab3fd Simplify addition and subtraction implementation
10  * 387f12 First version
```

Make a copy of that commit hash.

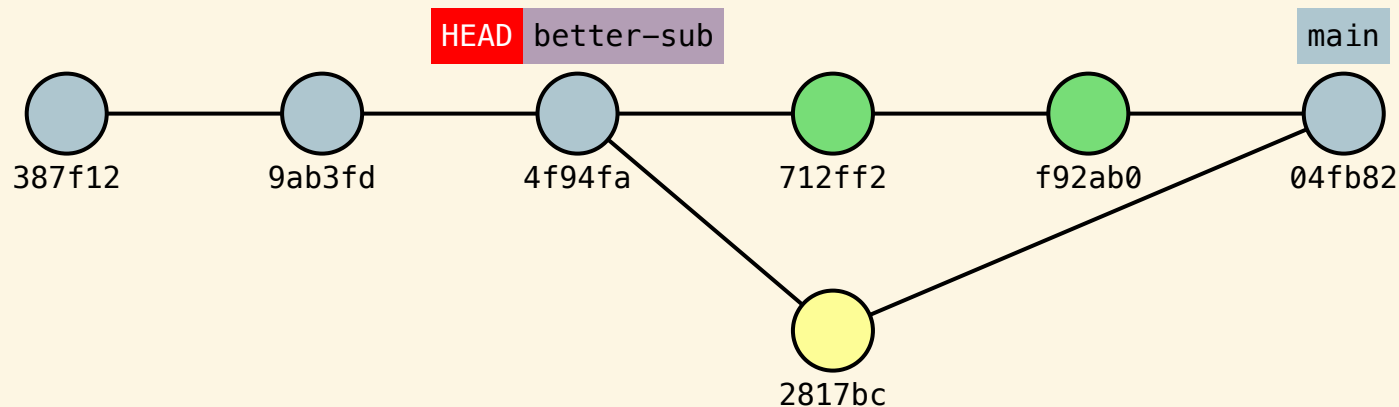
Speaker notes

We want to make it look as if your colleague did his work **at the same time** as you.

Note that the actual hash of the commit on your machine may be different than the one in this slide.

CREATE A BRANCH "IN THE PAST"

```
$> git switch -c better-sub 4f94fa  
# or git checkout -b better-sub 4f94fa
```



Speaker notes

You can create a branch at any point in the project's history by passing an additional commit reference to `git switch` or `git checkout`.

The `HEAD` has now moved to that point in the project's past history.

MAKE A CONFLICTING CHANGE



Now edit `subtraction.js` and implement subtraction again, but in a different way.

```
function subtract(a, b) {  
  return -b + a;  
}
```

CANNOT CHECK OUT CONFLICTING CHANGES

Git will not let you switch to `main` at this point:

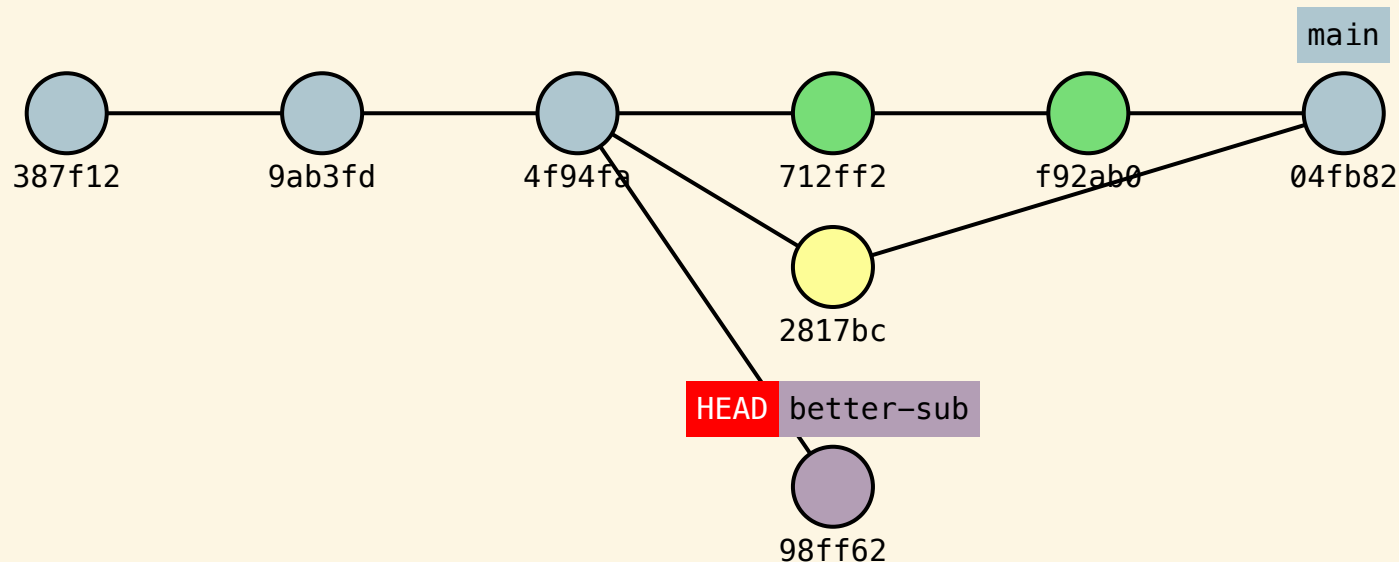
```
$> git switch main # or git checkout main
error: Your local changes to the following files would be
overwritten by checkout:
  subtraction.js
Please commit your changes or stash them before you
switch branches.
Aborting
```


Speaker notes

Git won't let you do it because the state of `subtraction.js` is different in that branch.

COMMIT THE CONFLICTING CHANGES

```
$> git add subtraction.js  
$> git commit -m "Implement a better subtract"
```



Speaker notes

Viewing the graph of commits, it's clear that the change has been made **in parallel** with our earlier changes.

MERGE THE CONFLICTING BRANCH

Go back to `main` and try to merge the `better-sub` branch:

```
$> git switch main # or git checkout main
$> git merge better-sub
Auto-merging subtraction.js
CONFLICT (content): Merge conflict in subtraction.js
Recorded preimage for 'subtraction.js'
Automatic merge failed; fix conflicts
and then commit the result.
```

It will fail!

Speaker notes

Git tells you that a **content conflict** has occurred in `subtraction.js`.

The merge has failed and no new commit has been created.

CHECK THE STATUS OF THE CONFLICT

Let's see what `git status` tells us:

```
$> git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   subtraction.js

no changes added to commit
  (use "git add" and/or "git commit -a")
```

Speaker notes

- Git tells you that the merge is **not complete**:
 - You can either fix the conflicts and run `git commit` to end the merge, or cancel the whole thing with `git merge --abort`.
- `subtraction.js` was modified in **both the current branch and the branch we are trying to merge in**.
- You can use `git add` to **mark the conflicts in a file as resolved**.

INSPECT THE CONFLICTED FILE

Let's see what's in `subtraction.js`:

```
/**
 * Takes two numbers a and b, and returns
 * the result of subtracting b from a.
 */
function subtract(a, b) {
<<<<<<< HEAD
  return a - b;
=====
  return -b + a;
>>>>>>> better-sub
}

calculate('subtraction', subtract);
```


Speaker notes

Notice two things here:

- Git has **successfully merged the comment** on the subtract function, since only one person changed these lines.
- Git could not merge the line with the computation, because the changes in the two branches conflict. It has added **conflict markers** to help you solve the issue.

CONFLICT MARKERS

Git has no idea what's right:

```
<<<<<< HEAD
    return a - b;
=====
    return -b + a;
>>>>>> better-sub
```

It is your responsibility to choose the correct version
(and remove the conflict markers).

Speaker notes

Take a closer look at the conflict markers:

- The section between <<<<<< HEAD and ===== is the content that was present in the current branch (HEAD) before you merged.
- The section between ===== and >>>>>> better-sub is the content that is being merged in from the better-sub branch.

Since Git cannot know which is better, it's **your responsibility** to:

- Remove the version you don't want, and...
- Remove the marker conflicts.

```
return -b + a;
```

Note that you could also write a new version combining changes from the two versions.

MARK THE CONFLICT AS RESOLVED

Now that you have fixed the conflict, do as instructed by Git and add the file to the staging area:

```
$> git add subtraction.js

$> git status
On branch main
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

      modified:   subtraction.js
```

COMMIT THE RESOLVED CONFLICTS

You still need to **commit** to end the merge:

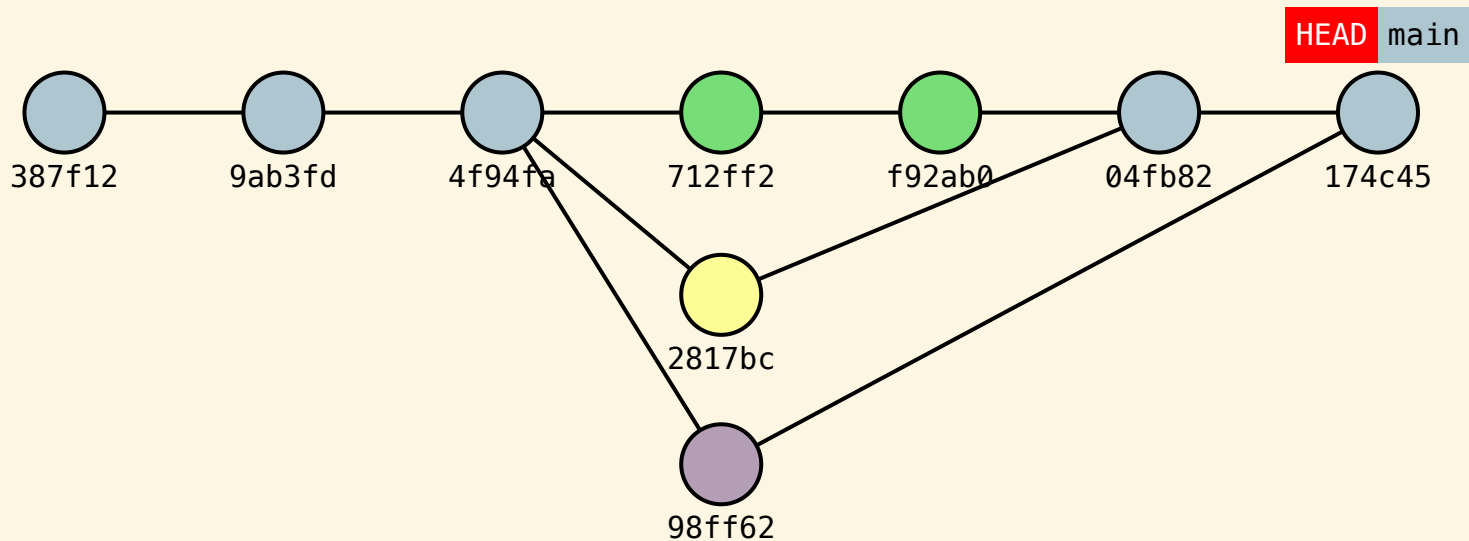
```
$> git commit -m "Merge better-sub into main"
```

Speaker notes

If you do not specify a commit message with `-m`, Git will generate one for you and open the configured editor (Vim by default) for you to check and/or change the message. Type `:wq` to exit from Vim or `Ctrl-X` to exit from nano, and to make the commit.

THE STATE AFTER MERGING

```
$> git branch -d better-sub
```



Speaker notes

The latest commit on `main` now includes the changes from all lines of development.

MERGE FILE CONFLICTS

Sometimes it's not just the contents of a file:

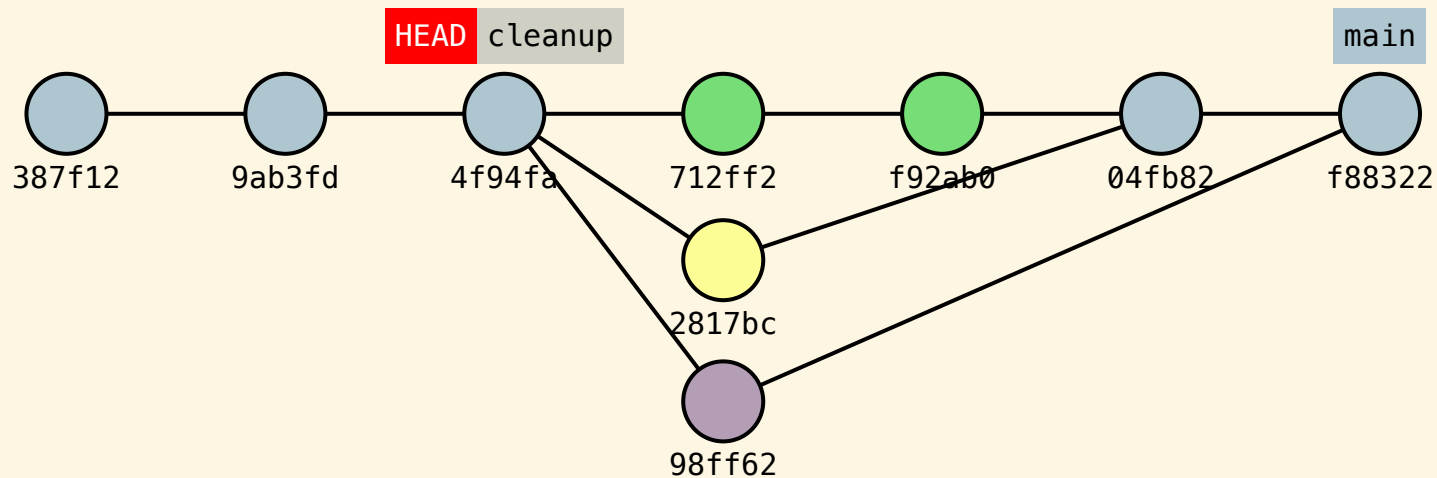
- You could have **modified a file** in your branch.
- *Someone else* could have **deleted it** in another branch.



It must have been someone else... right?

BACK TO THE ~~FUTURE~~ PAST

```
# or git checkout -b cleanup 4f94fa  
$> git switch -c cleanup 4f94fa
```

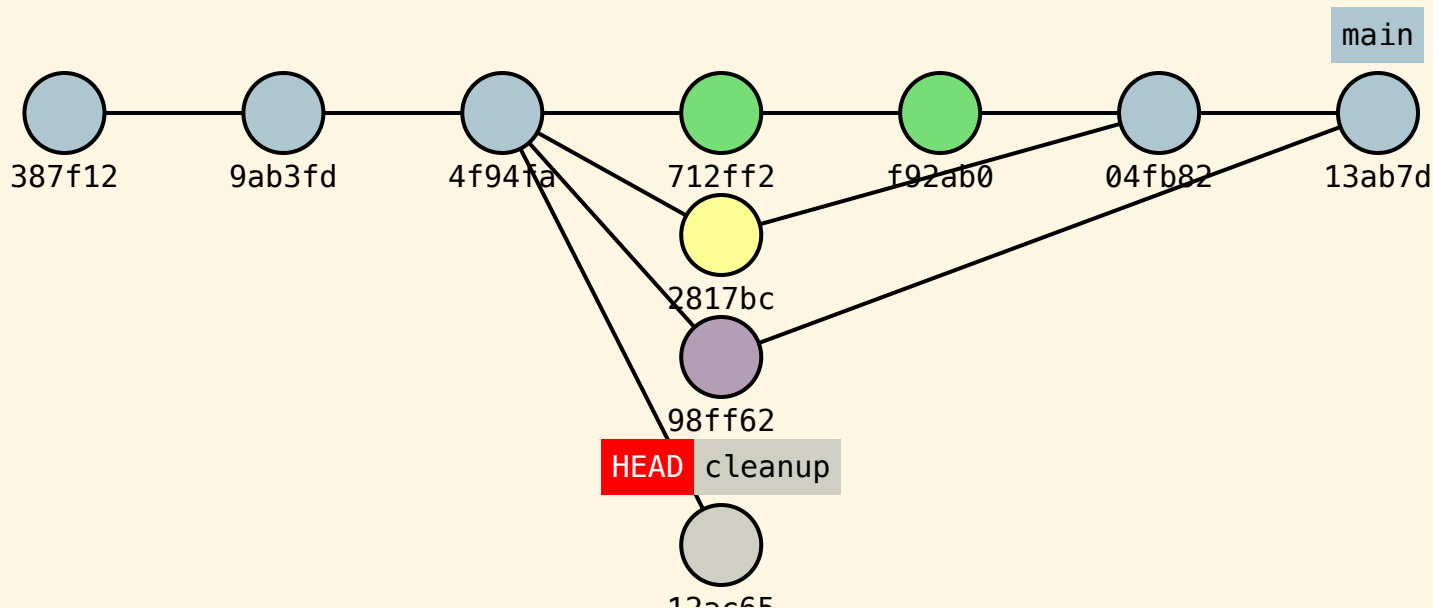


MAKE A CONFLICTING FILE CHANGE



This time, delete `subtraction.js`. We don't tolerate incomplete code in our project.

```
$> rm subtraction.js  
$> git add .  
$> git commit -m "Remove incomplete implementation"
```



MERGE THE CONFLICTING BRANCH

Let's try to merge that branch into `main`:

```
$> git switch main # or git checkout main
$> git merge cleanup
CONFLICT (modify/delete): subtraction.js deleted in cleanup
and modified in HEAD. Version HEAD of subtraction.js left
in tree.
Automatic merge failed; fix conflicts
and then commit the result.
```

Conflict!

Speaker notes

Git tells you immediately that there is a conflict and that:

- `subtraction.js` was **deleted** in the `cleanup` branch.
- `subtraction.js` was **modified** in the current branch (HEAD).
- Git **doesn't know** whether it should apply the deletion or the modification, so it left the modified file for you to check.

CHECK THE STATUS OF THE FILE CONFLICT

```
$> git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add/rm <file>..." as appropriate to mark resolution)

        deleted by them: subtraction.js

no changes added to commit
  (use "git add" and/or "git commit -a")
```

Speaker notes

Again, Git gives us some information:

- `subtraction.js` was **deleted by "them"**, meaning that it was deleted in the branch you're trying to merge in (if it had been deleted in the current branch and modified in the other branch, it would be *deleted by "us"*).
- Use either `git add` or `git rm` to mark the conflict as resolved.

TO DELETE, OR NOT TO DELETE...

You have to choose whether you want to either:

- **Keep** the modified file (use `git add`), or...
- **Remove** it (use `git rm`)

RESOLVE THE FILE CONFLICT

Let's keep it:

```
$> git add subtraction.js
$> git status
On branch main
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
```

COMMIT THE RESOLVED FILE CONFLICT

As instructed, use `git commit` to complete the merge:

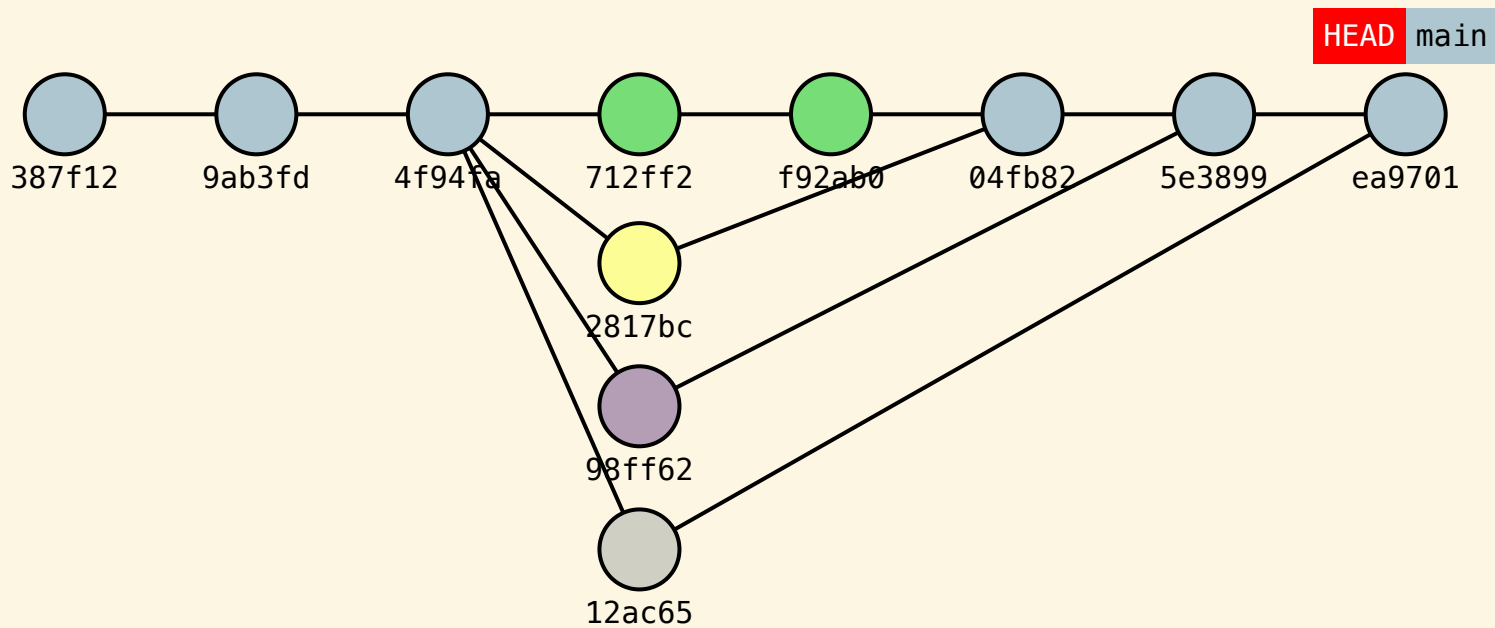
```
$> git commit -m "Merge cleanup (kept subtraction.js)"
```

Finally, delete the `cleanup` branch:

```
$> git branch -d cleanup
```

FINAL STATE

And you're done!



```
</file></file>
```