3-Way Merges in Git: Conflict Resolution and History Merges

IceCube Collaboration Meeting Spring 2021

Sebastian Fiedlschuster

https://github.com/fiedl/git-three-way-merges-talk
sebastian.fiedlschuster@fau.de
Slack: @fiedl

2021-03-19

Document 2021-siChoo2E

Erlangen Centre for Astroparticle Physics

Federal Ministry
of Education
and Research

Resources

New icetray repository on github:

`https://github.com/icecube/icetray`

Git migration:

`https://github.com/orgs/icecube/projects/1`

Repository with proposal, tests and experiments:

`https://github.com/fiedl/icecube-git-migration`

This talk:

`https://github.com/fiedl/git-three-way-merges-talk`

Other git talks:

`https://github.com/fiedl/icecube-git-migration-talk/releases`

Introduction

Why is it useful to think about merges?

- Automatic conflict resolution can save time
- Possible to track where things came from (implementation context)
- No downsides, except one needs to sometimes be extra careful when applying these tools.
  But: Can always try locally and push later because git can be used decentralized.

Introduction

What is a merge?

Oxford Advanced Learner’s Dictionary

to merge = to combine or make two or more things combine to form a single thing

https://www.oxfordlearnersdictionaries.com/definition/english/merge

git-scm

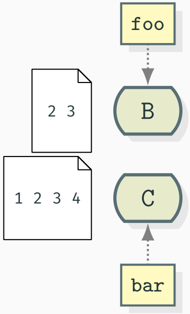
git-merge - Join two or more development histories together

Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch.

https://git-scm.com/docs/git-merge

Basic conflict resolution: 2-way merges (diffs)

Basic conflict resolution: 2-way merges (diffs)

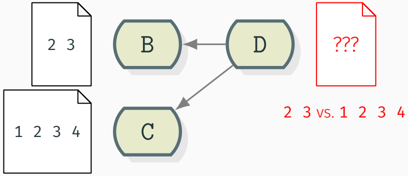


- Consider two unrelated versions (branches) of the same file

- What should be the merged version: 2 3 or 1 2 3 4 or something else? This a merge conflict.

Source: https://github.com/fiedl/git-three-way-merges-talk/issues/1

Basic conflict resolution: 2-way merges (diffs)

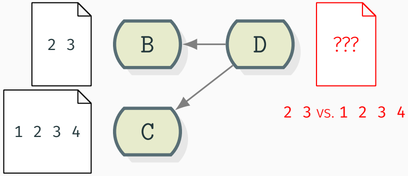


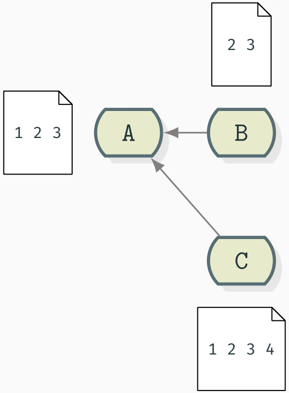
- Consider two unrelated versions (branches) of the same file
- What should be the merged version: 2 3 or 1 2 3 4 or something else? This a merge conflict.

Source: https://github.com/fiedl/git-three-way-merges-talk/issues/1

Basic conflict resolution: 3-way merges

Basic conflict resolution: 3-way merges



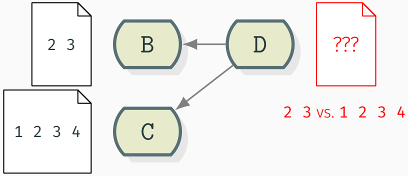


- Consider two unrelated versions (branches) of the same file
- What should be the merged version: 2 3 or 1 2 3 4 or something else? This a merge conflict.

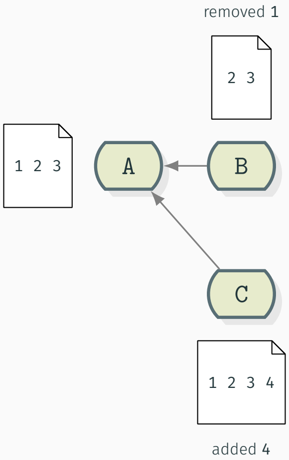
Source: https://github.com/fiedl/git-three-way-merges-talk/issues/2

- Consider two versions (branches) of the same file based on a common ancestor.
- As git knows the changes in comparison to the common ancestor …
- it can resolve the merge conflict automatically.

Basic conflict resolution: 3-way merges



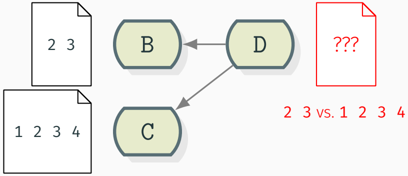
- Consider two unrelated versions (branches) of the same file
- What should be the merged version: 2 3 or 1 2 3 4 or something else? This a merge conflict.

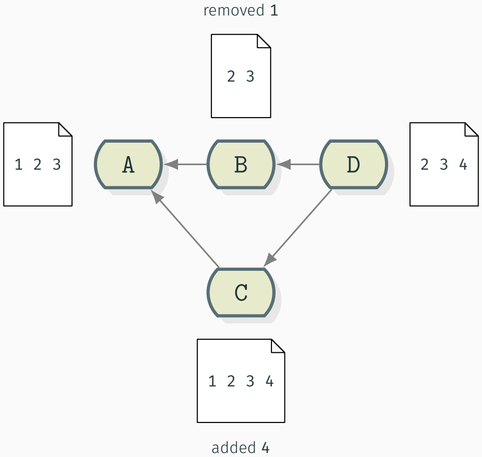


- Consider two versions (branches) of the same file based on a common ancestor.
- As git knows the changes in comparison to the common ancestor …
- it can resolve the merge conflict automatically.

Source: https://github.com/fiedl/git-three-way-merges-talk/issues/2

Basic conflict resolution: 3-way merges





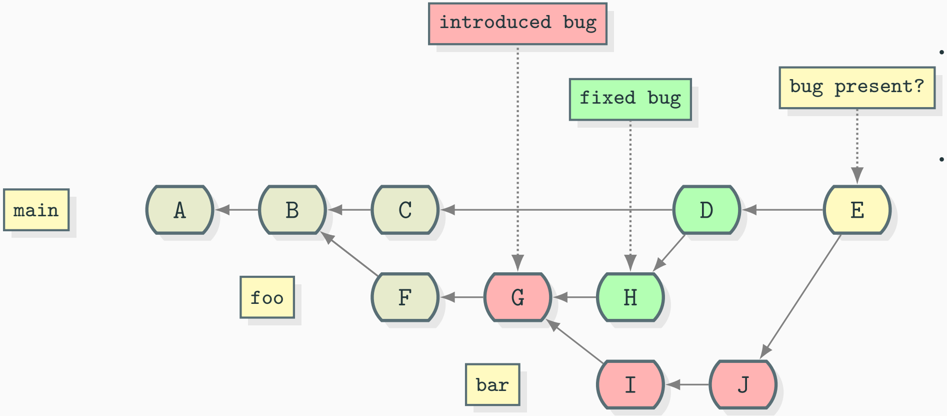
- Consider two unrelated versions (branches) of the same file
- What should be the merged version: 2 3 or 1 2 3 4 or something else? This a merge conflict.

- Consider two versions (branches) of the same file based on a common ancestor.
- As git knows the changes in comparison to the common ancestor ...
- it can resolve the merge conflict automatically.

Source: https://github.com/fiedl/git-three-way-merges-talk/issues/2

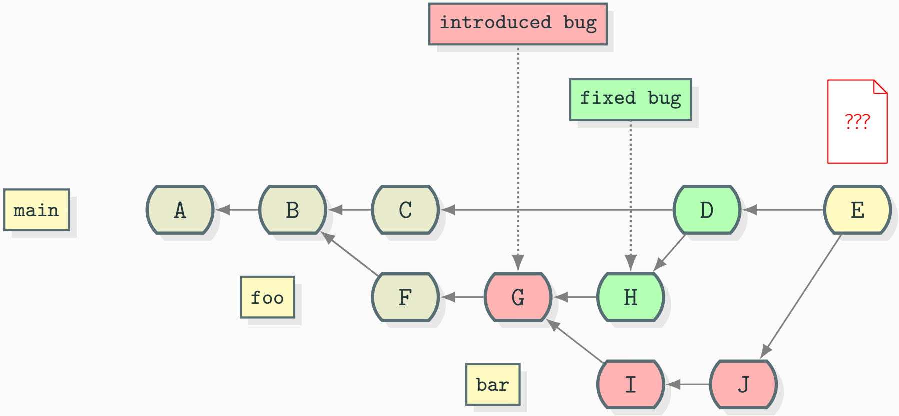
Example: Reintroduction of a fixed bug

Example: Reintroduction of a fixed bug



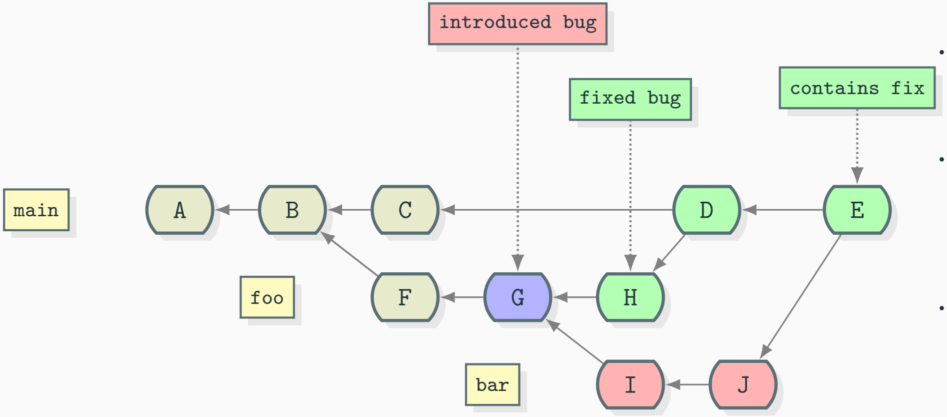
- Consider a feature branch `foo` that introduces a bug (`G`)
- The bug gets fixed during review (`H`)
- However, you base work on the buggy code while waiting for review (`I`, `J`)
- When merging this additional work into `main` (`E`), does it contain the bug or the fix?

Example: Reintroduction of a fixed bug (2-way merges)



- Consider a feature branch `foo` that introduces a bug (`G`)
- The bug gets fixed during review (`H`)
- However, you base work on the buggy code while waiting for review (`I`, `J`)
- When merging this additional work into `main` (`E`), does it contain the bug or the fix?
- Answer: With 2-way merges, git can't know and will present a merge conflict.

Example: Reintroduction of a fixed bug (3-way merges)

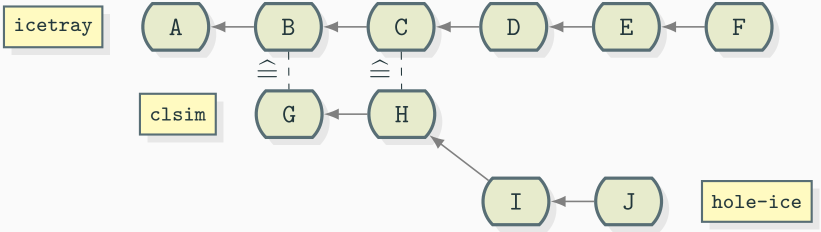


Source: https://github.com/fiedl/icecube-git-migration/issues/15

- Consider a feature branch `foo` that introduces a bug (`G`)
- The bug gets fixed during review (`H`)
- However, you base work on the buggy code while waiting for review (`I`, `J`)
- When merging this additional work into `main` (`E`), does it contain the bug or the fix?
- Answer: With 2-way merges, git can't know and will present a merge conflict.
- With 3-way merges, git can identify the common ancestor (`G`) and resolve the conflict automatically. `E` contains the fix.

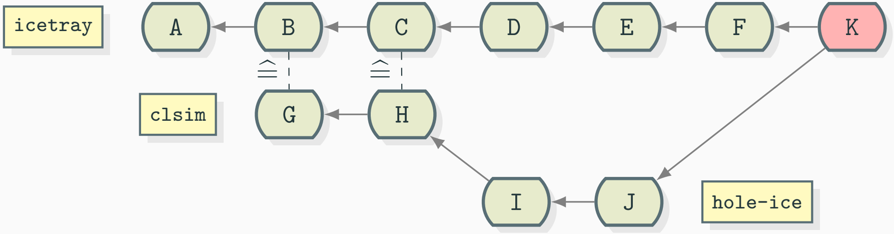
Application: Merge feature from other history

Application: Merge feature from other history



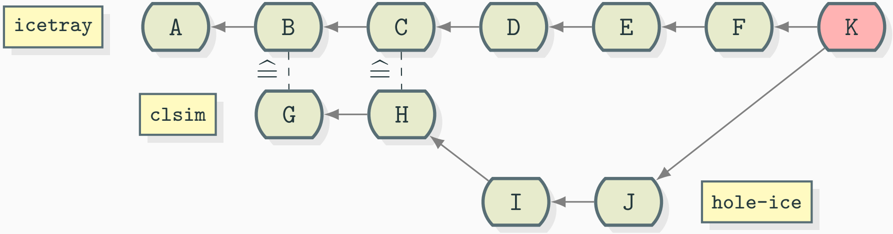
- Consider a feature (`hole-ice`) based on other history (`clsim`) than the main one (`icetray`)
- The histories are mirrored, but have different commit ids (`B` ≙ `G`, `C` ≙ `H`)
- Want to merge `hole-ice` into `icetray`
- Without a common ancestor, this is a 2-way merge and all merge conflicts must be resolved manually
- Need to connect histories first in order to create a common ancestor

Application: Merge feature from other history



- Consider a feature (`hole-ice`) based on other history (`clsim`) than the main one (`icetray`)
- The histories are mirrored, but have different commit ids (`B` ≙ `G`, `C` ≙ `H`)
- Want to merge `hole-ice` into `icetray`
- Without a common ancestor, this is a 2-way merge and all merge conflicts must be resolved manually
- Need to connect histories first in order to create a common ancestor

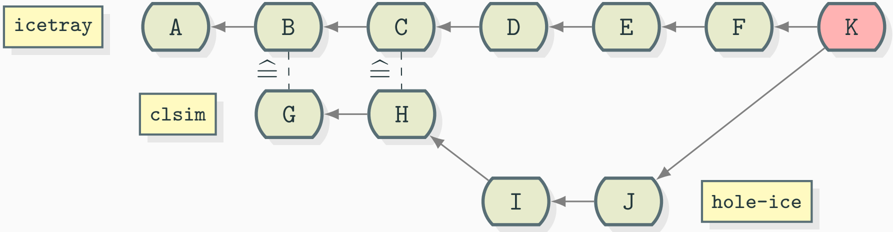
Application: Merge feature from other history



```
git checkout icetray
git merge --allow-unrelated-histories hole-ice
```

- Consider a feature (`hole-ice`) based on other history (`clsim`) than the main one (`icetray`)
- The histories are mirrored, but have different commit ids (`B` ≙ `G`, `C` ≙ `H`)
- Want to merge `hole-ice` into `icetray`
- Without a common ancestor, this is a 2-way merge and all merge conflicts must be resolved manually
- Need to connect histories first in order to create a common ancestor

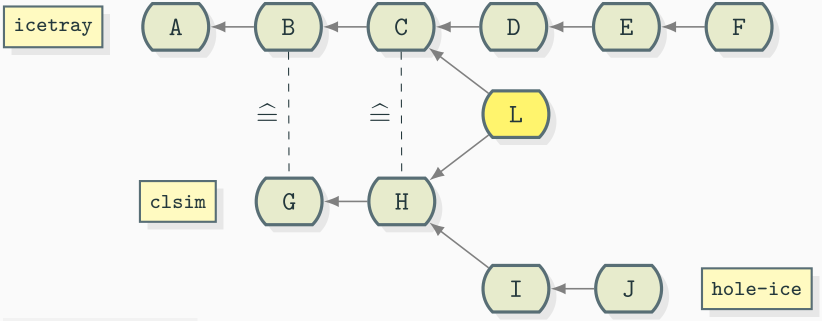
Application: Merge feature from other history



```
git checkout icetray
git merge --allow-unrelated-histories hole-ice
```

- Consider a feature (`hole-ice`) based on other history (`clsim`) than the main one (`icetray`)
- The histories are mirrored, but have different commit ids (`B` ≙ `G`, `C` ≙ `H`)
- Want to merge `hole-ice` into `icetray`
- Without a common ancestor, this is a 2-way merge and all merge conflicts must be resolved manually
- Need to connect histories first in order to create a common ancestor

Application: Merge feature from other history

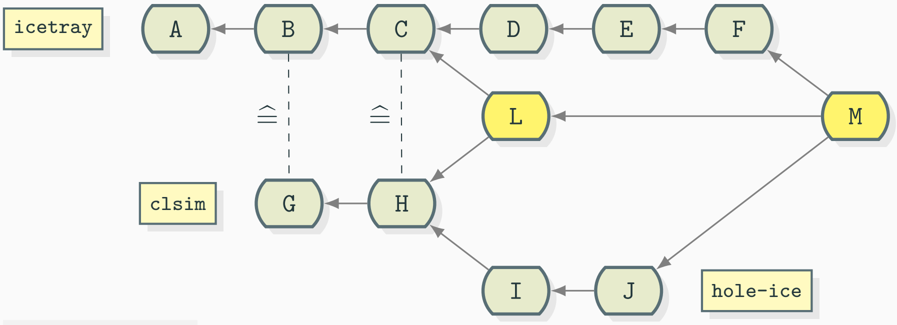


```
git checkout C
git merge --allow-unrelated-histories --strategy=ours --no-ff H
```

- Connect both histories and identify two corresponding commits (`C` ≙ `H` ≙ `L`)
- Bring in changes of `icetray` and `hole-ice`: `M` contains current `icetray` as well as `hole-ice`.
- Merge conflicts are resolved automatically as git knows the common ancestors (`C` for merging `F`, `H` for merging `J`)
- To see a nice diff and use review tools, create a pull request for `M` against `icetray`.

Source: https://github.com/fiedl/icecube-git-migration/issues/14

Application: Merge feature from other history

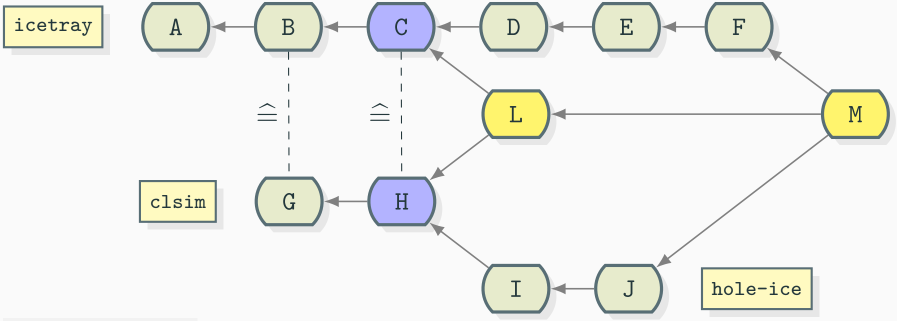


```
git checkout C
git merge --allow-unrelated-histories --strategy=ours --no-ff H
git merge icetray hole-ice
```

- Connect both histories and identify two corresponding commits (`C` ≙ `H` ≙ `L`)
- Bring in changes of `icetray` and `hole-ice`: `M` contains current `icetray` as well as `hole-ice`.
- Merge conflicts are resolved automatically as git knows the common ancestors (`C` for merging `F`, `H` for merging `J`)
- To see a nice diff and use review tools, create a pull request for `M` against `icetray`.

Source: https://github.com/fiedl/icecube-git-migration/issues/14

Application: Merge feature from other history

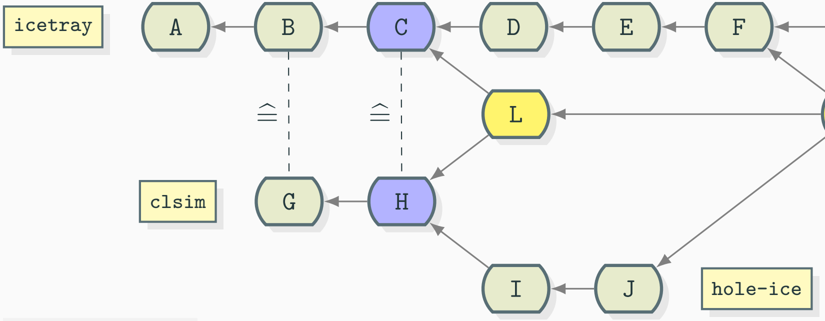


```
git checkout C
git merge --allow-unrelated-histories --strategy=ours --no-ff H
git merge icetray hole-ice
```

- Connect both histories and identify two corresponding commits (`C` ≙ `H` ≙ `L`)
- Bring in changes of `icetray` and `hole-ice` : `M` contains current `icetray` as well as `hole-ice` .
- Merge conflicts are resolved automatically as git knows the common ancestors (`C` for merging `F` , `H` for merging `J`)
- To see a nice diff and use review tools, create a pull request for `M` against `icetray` .

Source: https://github.com/fiedl/icecube-git-migration/issues/14

Application: Merge feature from other history



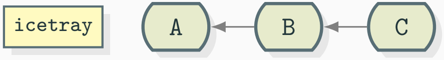
```
git checkout C
git merge --allow-unrelated-histories --strategy=ours --no-ff H
git merge icetray hole-ice
```

- Connect both histories and identify two corresponding commits (`C` ≙ `H` ≙ `L`)
- Bring in changes of `icetray` and `hole-ice`: `M` contains current `icetray` as well as `hole-ice`.
- Merge conflicts are resolved automatically as git knows the common ancestors (`C` for merging `F`, `H` for merging `J`)
- To see a nice diff and use review tools, create a pull request for `M` against `icetray`.

Source: https://github.com/fiedl/icecube-git-migration/issues/14

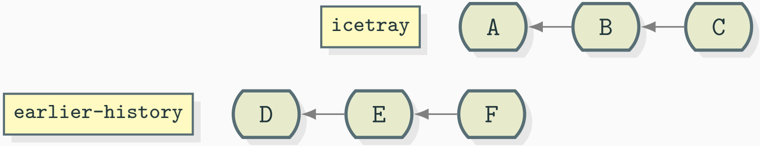
Application: Bring in earlier history

Application: Bring in earlier history



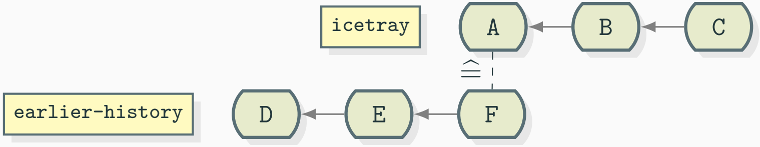
- Consider the history of `icetray` imported from svn beginning at an arbitrary revision
- Later, we might import earlier history from svn
- `A` and `F` correspond to the same svn revision, but have different commit ids due to the separate imports
- Now we want to merge the `earlier-history` into `icetray`
- But as there is no common ancestor, all merge conflicts need to be resolved manually

Application: Bring in earlier history



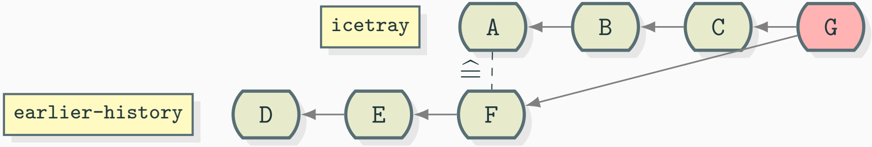
- Consider the history of `icetray` imported from svn beginning at an arbitrary revision
- Later, we might import earlier history from svn
- `A` and `F` correspond to the same svn revision, but have different commit ids due to the separate imports
- Now we want to merge the `earlier-history` into `icetray`
- But as there is no common ancestor, all merge conflicts need to be resolved manually

Application: Bring in earlier history



- Consider the history of `icetray` imported from svn beginning at an arbitrary revision
- Later, we might import earlier history from svn
- `A` and `F` correspond to the same svn revision, but have different commit ids due to the separate imports
- Now we want to merge the `earlier-history` into `icetray`
- But as there is no common ancestor, all merge conflicts need to be resolved manually

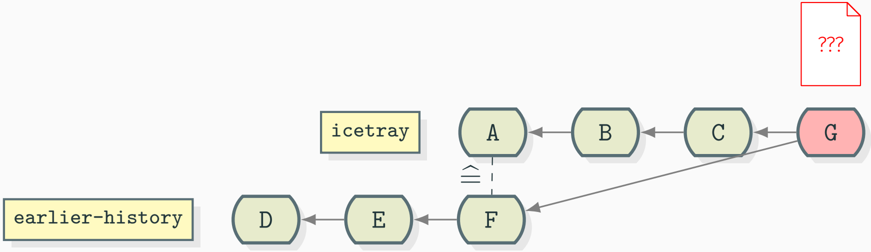
Application: Bring in earlier history



```
git checkout icetray
git merge --allow-unrelated-histories --no-ff earlier-history
```

- Consider the history of `icetray` imported from svn beginning at an arbitrary revision
- Later, we might import earlier history from svn
- `A` and `F` correspond to the same svn revision, but have different commit ids due to the separate imports
- Now we want to merge the `earlier-history` into `icetray`
- But as there is no common ancestor, all merge conflicts need to be resolved manually

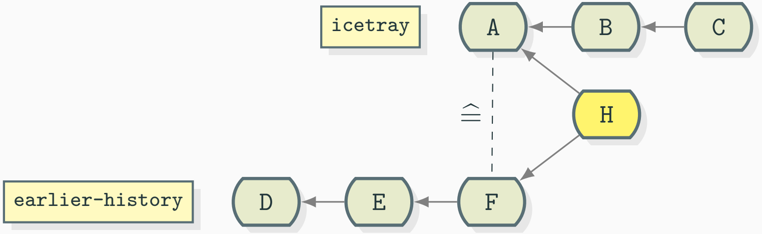
Application: Bring in earlier history



```
git checkout icetray
git merge --allow-unrelated-histories --no-ff earlier-history
```

- Consider the history of `icetray` imported from svn beginning at an arbitrary revision
- Later, we might import earlier history from svn
- `A` and `F` correspond to the same svn revision, but have different commit ids due to the separate imports
- Now we want to merge the `earlier-history` into `icetray`
- But as there is no common ancestor, all merge conflicts need to be resolved manually

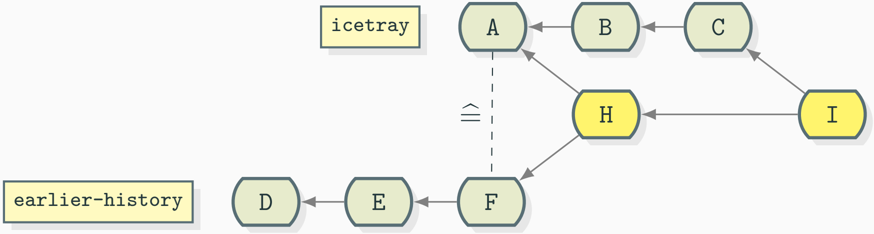
Application: Bring in earlier history



```
git checkout A
git merge --allow-unrelated-histories --strategy=ours --no-ff F
```

- Connect both histories and identify corresponding commits (`A` ≙ `F` ≙ `H`)
- Bring in newer commits
- Create a pull request for `I` against `icetray` for convenience, e.g. to document the process

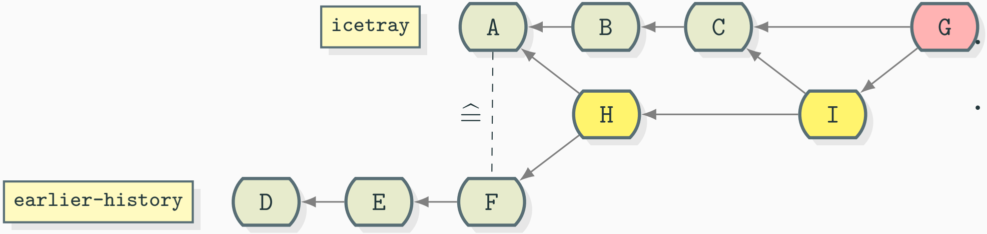
Application: Bring in earlier history



```
git checkout A
git merge --allow-unrelated-histories --strategy=ours --no-ff F
git merge --no-ff C
```

- Connect both histories and identify corresponding commits (`A` ≙ `F` ≙ `H`)
- Bring in newer commits
- Create a pull request for `I` against `icetray` for convenience, e.g. to document the process

Application: Bring in earlier history

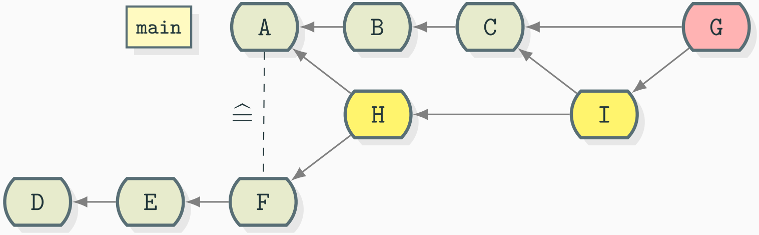


- Connect both histories and identify corresponding commits (`A` $\widehat{=}$ `F` $\widehat{=}$ `H`)
- Bring in newer commits
- Create a pull request for `I` against `icetray` for convenience, e.g. to document the process

```
git checkout A
git merge --allow-unrelated-histories --strategy=ours --no-ff F
git merge --no-ff C
```

Useful tricks

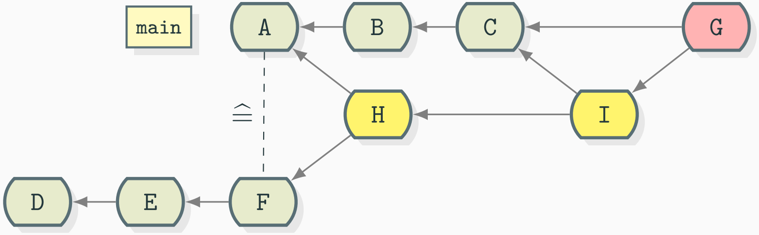
Trick: `git log` only history of `main`



By default:

```
$ git checkout main
$ git log
G import earlier history
I merging main into temp
C blub
H merging F into temp
B blah
F baz
A baz
E bar
D foo
```

Trick: `git log` only history of `main`



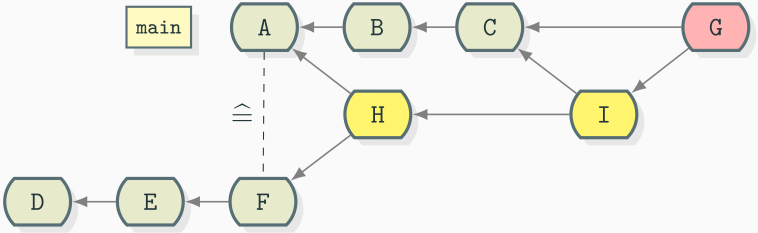
More conveniently:

```
$ git checkout main
$ git log --first-parent
G import earlier history
C blub
B blah
A baz
```

With alias:

```
$ git config --
global alias.lg "log --first-
parent"
$ git lg
```

Trick: `git graph`

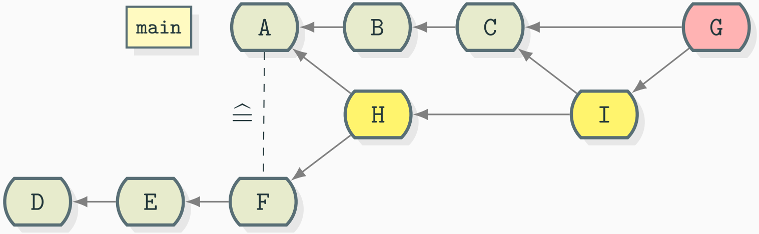


Show the graph in the command line:

```
$ git checkout main
$ git log --decorate --oneline --
graph
*   G (HEAD -> main) import earlier h
|\
| *   I (temp) merging main into temp
| |\
| |/
|/|
* | C blub
* | B blah
| * H merging F into temp
|/|
| * (earlier-history) F baz
| * E bar
| * D foo
* A baz
```

See also: `https://stackoverflow.com/q/1057564/2066546`, `https://github.com/fiedl/git-three-way-merges-talk/issues/3`

Trick: `git graph`

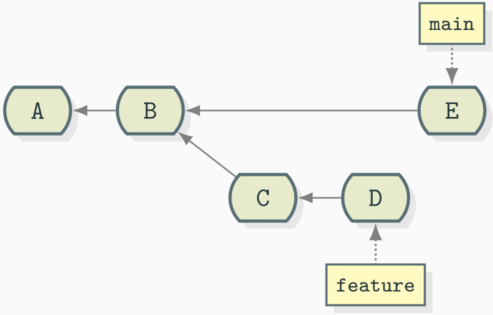


Define `git graph` as alias:

```
$ git config \
  --global alias.graph \
  "log --decorate --oneline --
graph"

$ git graph
*   G (HEAD -> main) import earlier h
|\
| *   I (temp) merging main into temp
| |\
...
```

See also: https://stackoverflow.com/q/1057564/2066546, https://github.com/fiedl/git-three-way-merges-talk/issues/3

Cave: Avoid renaming the `main` branch



What is renaming the `main` branch?

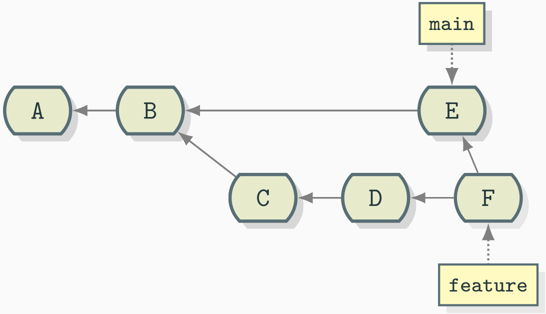
- Consider a `main` branch and a `feature` branch.
- The author of the `feature` branch pulls in the `main` branch to get the changes that have been committed to the `main` branch in the meantime:

```
git checkout feature
git merge main
```

- The author of the `feature` branch merges his feature branch in to `main`, which would be a fast-forward merge at this point.

```
git checkout main
git merge feature
```

Cave: Avoid renaming the `main` branch



What is renaming the `main` branch?

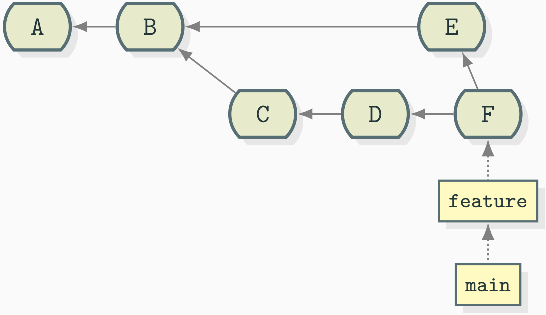
- Consider a `main` branch and a `feature` branch.
- The author of the `feature` branch pulls in the `main` branch to get the changes that have been committed to the `main` branch in the meantime:

```
git checkout feature
git merge main
```

- The author of the `feature` branch merges his feature branch in to `main`, which would be a fast-forward merge at this point.

```
git checkout main
git merge feature
```

Cave: Avoid renaming the `main` branch



What is renaming the `main` branch?

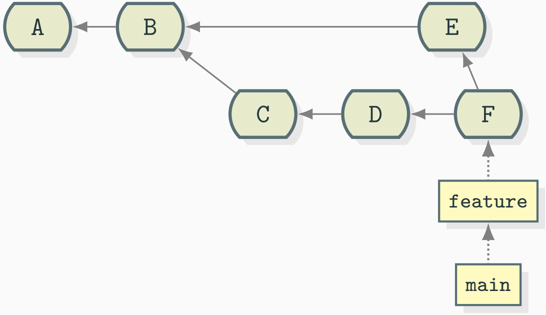
- Consider a `main` branch and a `feature` branch.
- The author of the `feature` branch pulls in the `main` branch to get the changes that have been committed to the `main` branch in the meantime:

```
git checkout feature
git merge main
```

- The author of the `feature` branch merges his feature branch in to `main`, which would be a fast-forward merge at this point.

```
git checkout main
git merge feature
```

Cave: Avoid renaming the `main` branch



The `main` branch now contains the commits `A B C D F` rather than `A B E`.

This breaks bisection and readable changelogs.

What is renaming the `main` branch?

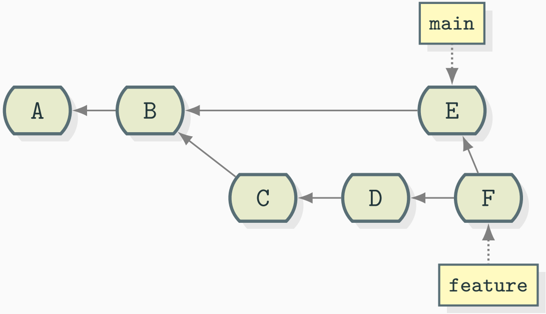
- Consider a `main` branch and a `feature` branch.
- The author of the `feature` branch pulls in the `main` branch to get the changes that have been committed to the `main` branch in the meantime:

```
git checkout feature
git merge main
```

- The author of the `feature` branch merges his feature branch in to `main`, which would be a fast-forward merge at this point.

```
git checkout main
git merge feature
```

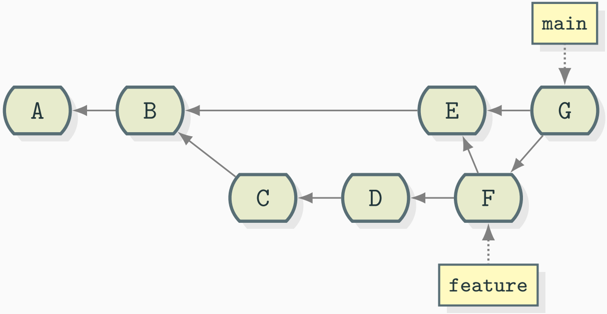
Cave: Avoid renaming the `main` branch



- Suppose you need to keep all commits of the feature branch because they are already pushed.
- Use `--no-ff` to avoid accidental fast-forward merges:

```
git checkout main
git merge --no-ff feature
```

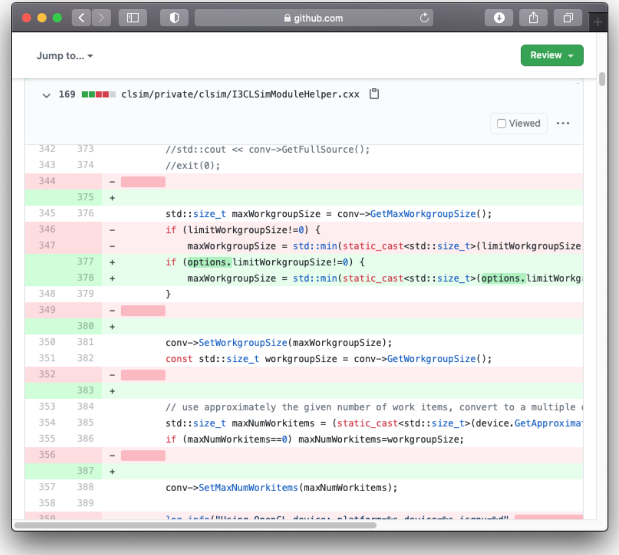
Cave: Avoid renaming the `main` branch



- Suppose you need to keep all commits of the feature branch because they are already pushed.
- Use `--no-ff` to avoid accidental fast-forward merges:

```
git checkout main
git merge --no-ff feature
```

Cave: Avoid renaming the `main` branch



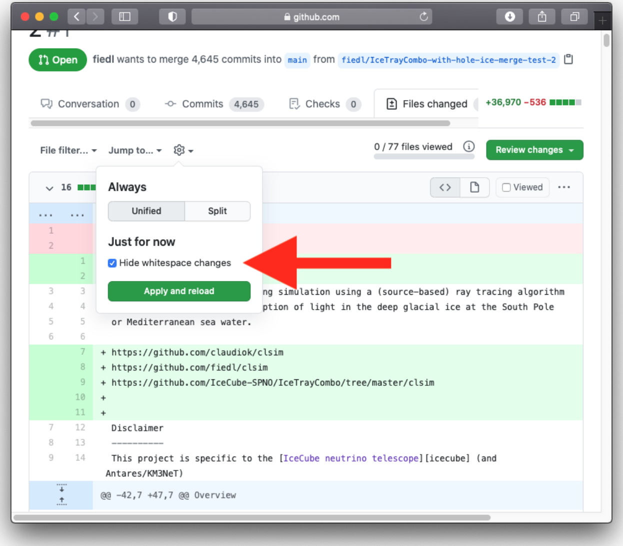
```
         //std::cout << conv->GetFullSource();
         //exit(0);
-
+
         std::size_t maxWorkgroupSize = conv->GetMaxWorkgroupSize();
-        if (limitWorkgroupSize!=0) {
-            maxWorkgroupSize = std::min(static_cast<std::size_t>(limitWorkgroupSize
+        if (options.limitWorkgroupSize!=0) {
+            maxWorkgroupSize = std::min(static_cast<std::size_t>(options.limitWorkg
         }
-
+
         conv->SetWorkgroupSize(maxWorkgroupSize);
         const std::size_t workgroupSize = conv->GetWorkgroupSize();
-
+
         // use approximately the given number of work items, convert to a multiple
         std::size_t maxNumWorkitems = (static_cast<std::size_t>(device.GetApproxima
         if (maxNumWorkitems==0) maxNumWorkitems=workgroupSize;
-
+
         conv->SetMaxNumWorkitems(maxNumWorkitems);

         log_info("Using OpenCL device: platform=%s device=%s isgpu=%d"
```

- Don’t assume that all developers will agree on one whitespace convention.
- But in diffs, it’s sometimes hard to see the actual changes due to whitespace clutter.
- Ignore whitespace in diffs and automatic conflict resolution:

```
git merge -X ignore-all-space ...
```

- This can also be used in pull requests and diffs on github by clicking Hide whitespace changes

Cave: Avoid renaming the `main` branch



Open fiedl wants to merge 4,645 commits into `main` from `fiedl/IceTrayCombo-with-hole-ice-merge-test-2`

☑ Hide whitespace changes

```
  ...ng simulation using a (source-based) ray tracing algorithm
  ...ption of light in the deep glacial ice at the South Pole
  or Mediterranean sea water.

+ https://github.com/claudiok/clsim
+ https://github.com/fiedl/clsim
+ https://github.com/IceCube-SPNO/IceTrayCombo/tree/master/clsim
+
+
  Disclaimer
  ----------
  This project is specific to the [IceCube neutrino telescope][icecube] (and
Antares/KM3NeT)
```

- Don't assume that all developers will agree on one whitespace convention.
- But in diffs, it's sometimes hard to see the actual changes due to whitespace clutter.
- Ignore whitespace in diffs and automatic conflict resolution:

  ```
  git merge -X ignore-all-space ...
  ```

- This can also be used in pull requests and diffs on github by clicking Hide whitespace changes

Summary

- Thoughtful merging can save time in the long run and helps keeping a long-term track of the context where changes came from
- Bring two equivalent git graph nodes (commits) *A* and *B* together with:
  `git co A`
  `git merge --allow-unrelated-histories --strategy=ours --no-ff -X ignore-all-space B`
- Use `strategy=ours` vs. `strategy=theirs` careful depending on which branch should be followed in `git blame`
- Be careful not to rename the `main` branch
- Be sure to go through a pull request to bring it into `main` in order to double-check that everything has worked as intended

Thanks for your attention!

Thanks for your attention!

Any input you might have is welcome:

`https://github.com/fiedl/icecube-git-migration/issues`

`sebastian.fiedlschuster@fau.de`

Slack: `@fiedl`