LEIDEN UNIVERSITY MEDICAL CENTER

Working with branches in Git

Martijn Vermaat

Leiden Genome Technology Center

Department of Human Genetics

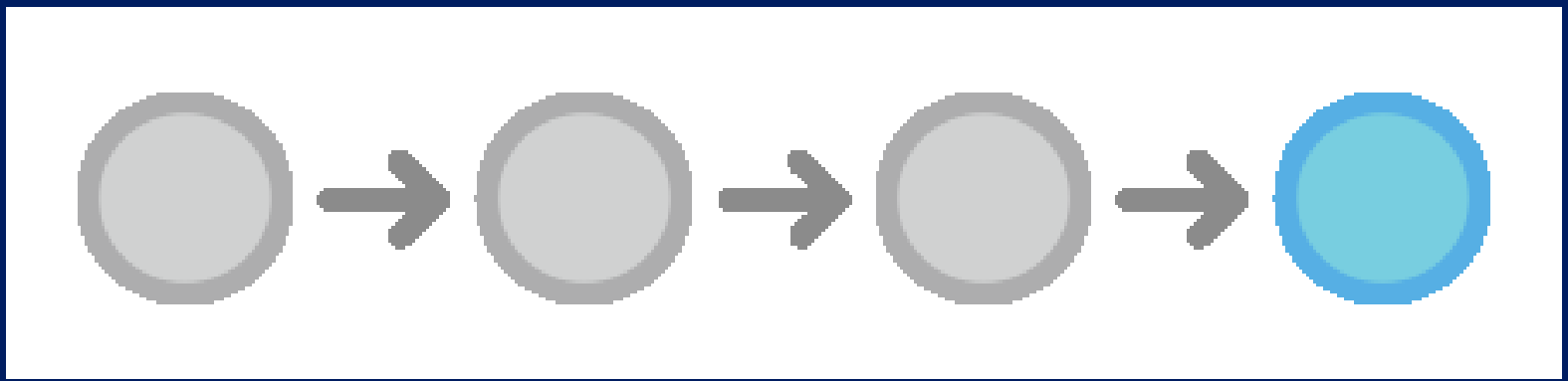
Center for Human and Clinical Genetics

The Git commit graph

Table of contents

The Git commit graph

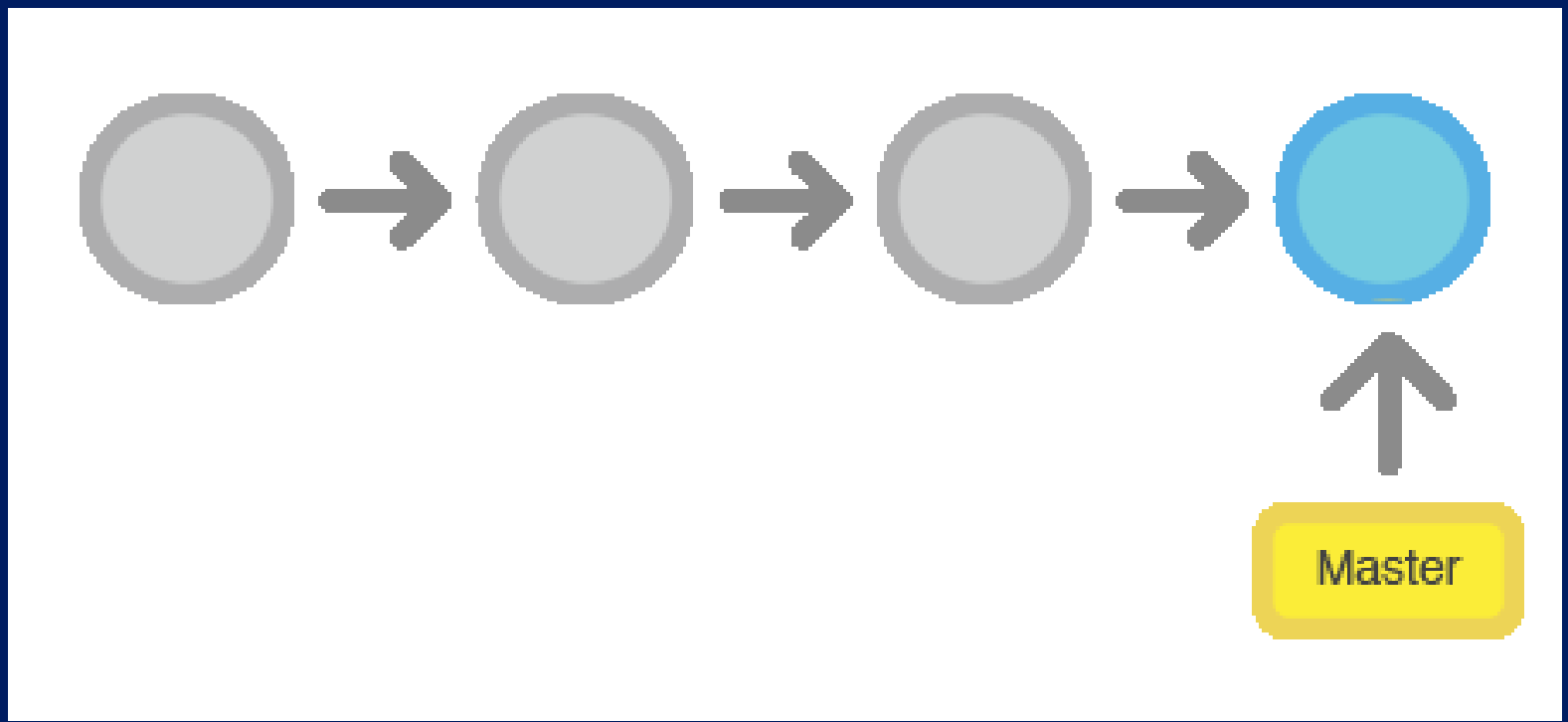
A linear history



- Every commit has a parent.
- *Committing* creates a new commit on top of the current one.

The Git commit graph

The default `master` branch

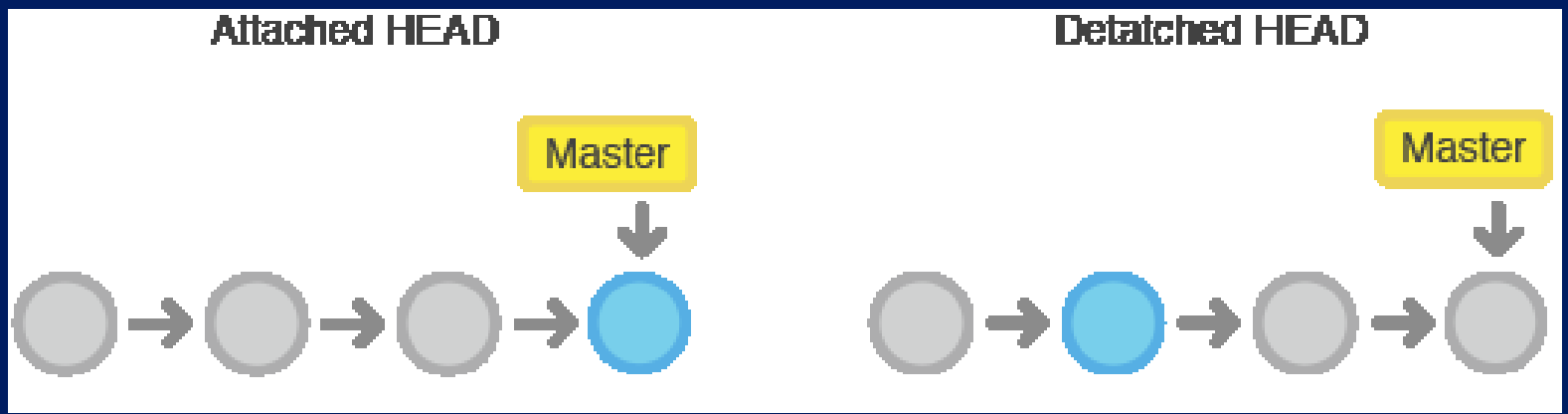


- A *branch* is a pointer to a commit.
- By default there is one branch: **`master`**.

The Git commit graph

The current commit: `HEAD`

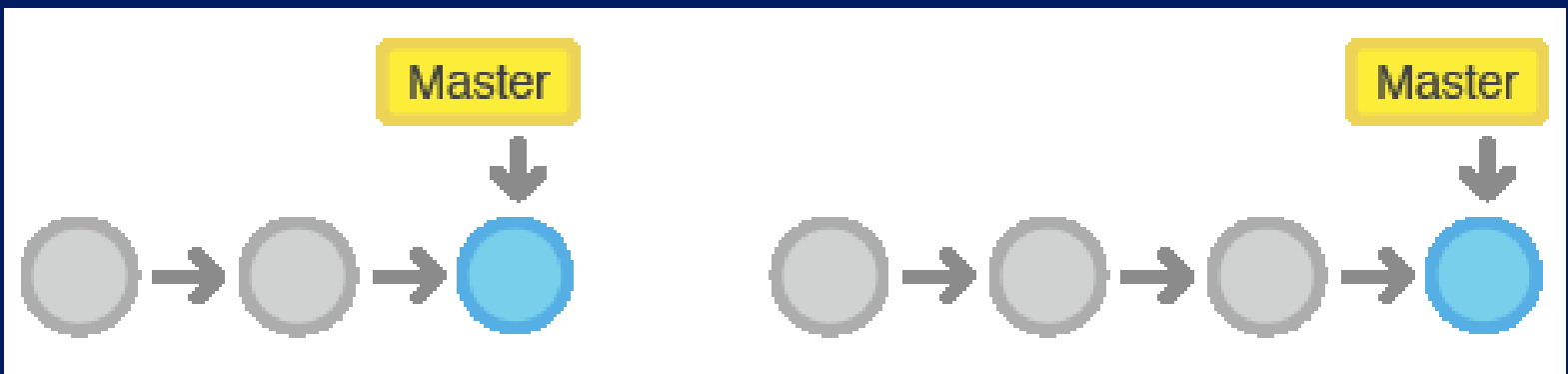
- The current commit is called **`HEAD`** (shown in blue).
- **`HEAD`** normally points to the current branch.
- Or to a commit if there is no current branch.



The Git commit graph

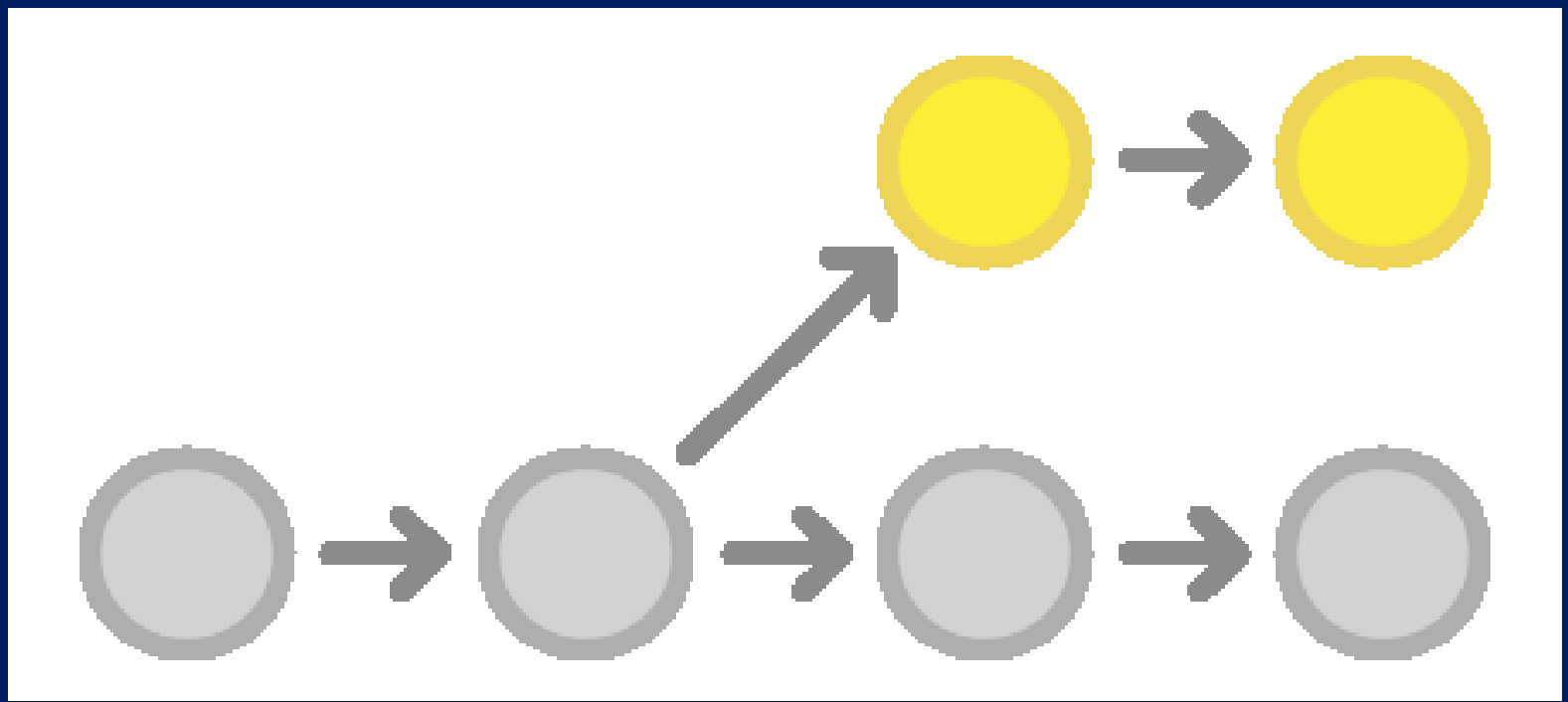
Committing moves `HEAD` and the current branch

- Committing moves the current branch to the new commit.
- Of course, **`HEAD`** moves with it.
- If there is no current branch, only **`HEAD`** moves.



The Git commit graph

A non-linear history

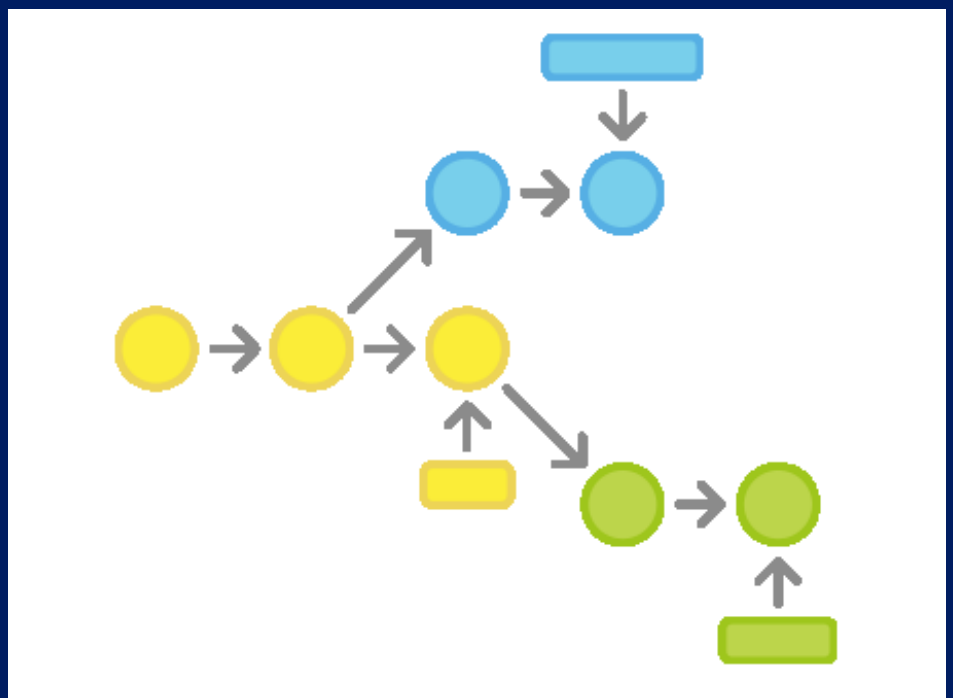


- The commit graph can become non-linear.
- Usually by committing from the same commit twice.

The Git commit graph

Use branches to keep track of different code paths

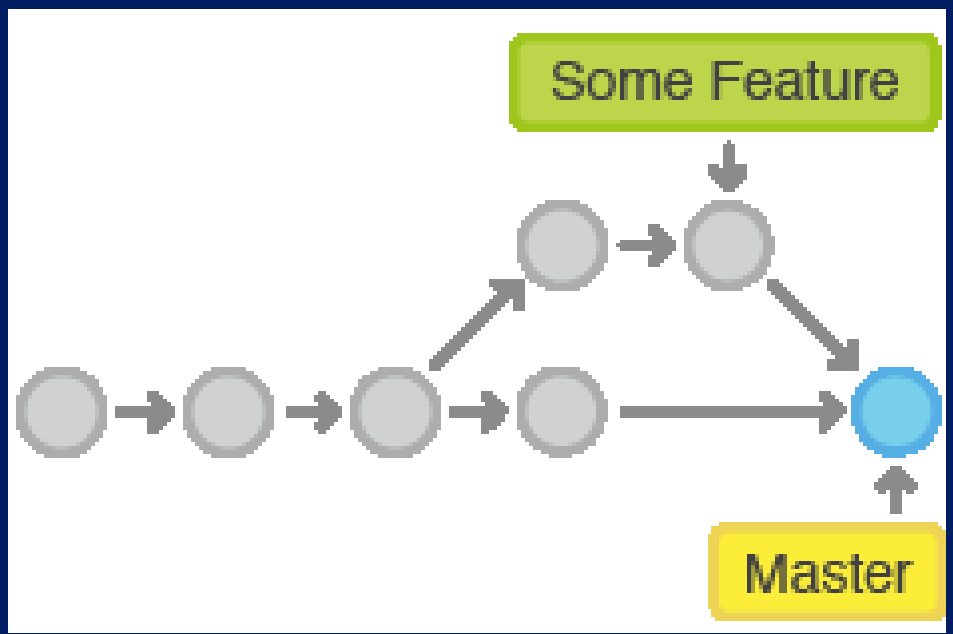
- Branch names are easier to remember than commit hashes.
- Branch names make it clear what commits are about.



The Git commit graph

Different code paths may later join

- Commits from different branches can be brought together.
- We call this *merging*.

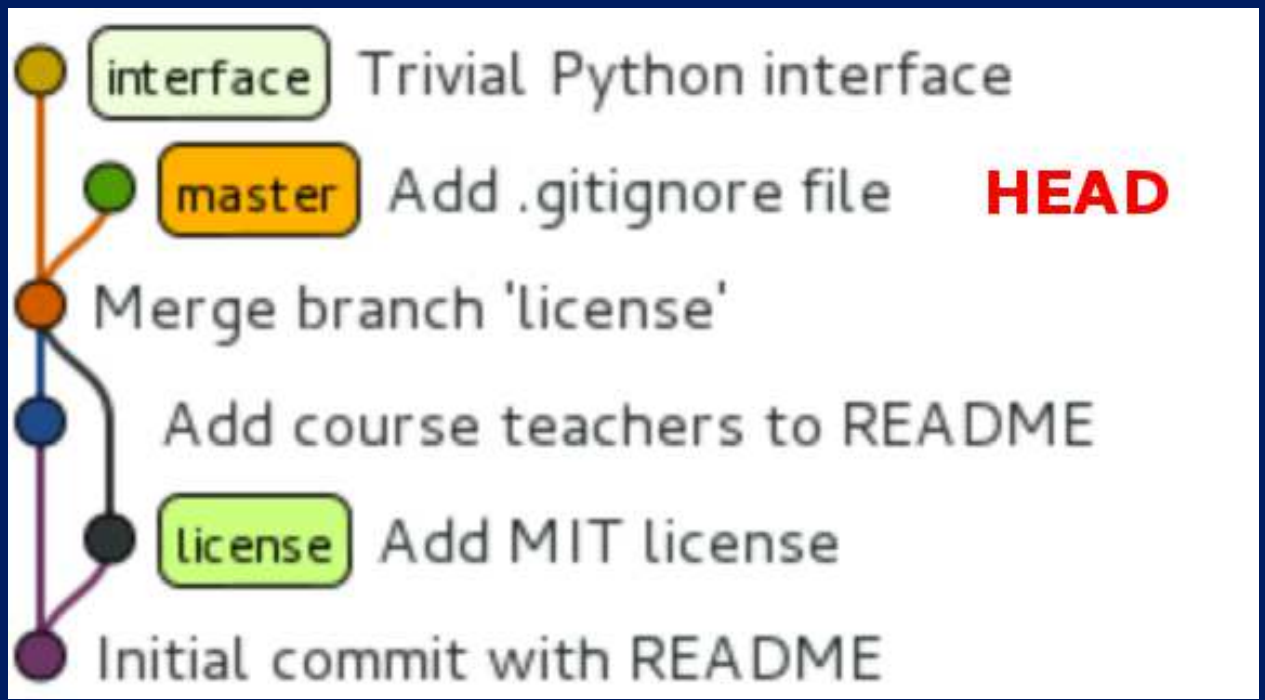


Inspecting the commit graph

Table of contents

Inspecting the commit graph

An example repository



- Branches **`license`** and **`interface`** diverged from **`master`**.
- Only **`license`** has been merged back into **`master`**.
- Current branch is **`master`**.

Inspecting the commit graph

Showing the current branch: `git status`

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Remember: you cannot type **`git status`** enough!

Inspecting the commit graph

Listing branches: `git branch`

```
$ git branch
  interface
  license
* master
```

Three branches, current branch is **`master`**.

Inspecting the commit graph

Listing branches: `git branch`

```
$ git branch
  interface
  license
* master
```

Three branches, current branch is **`master`**.

```
$ git branch -v
  interface 8fc25c1 Trivial Python interface
  license   0fbe3e3 Add MIT license
* master    c7f3bd9 Add .gitignore file
```

`-v`: Shows the commit each branch points to.

Inspecting the commit graph

The commit log: `git log`

```
$ git log --oneline --decorate
c7f3bd9 (HEAD, master) Add .gitignore file
4a44c4e Merge branch 'license'
64af1ee Add course teachers to README
0fbe3e3 (license) Add MIT license
d1c7fd7 Initial commit with README
```

`--oneline`: Shows commit summary on one line.
`--decorate`: Adds branch information.

Inspecting the commit graph

The commit log as a graph: `git log`

```
$ git log --oneline --decorate --graph --all
* 8fc25c1 (interface) Trivial Python interface
| * c7f3bd9 (HEAD, master) Add .gitignore file
|/
*   4a44c4e Merge branch 'license'
|\
| * 0fbe3e3 (license) Add MIT license
* | 64af1ee Add course teachers to README
|/
* d1c7fd7 Initial commit with README
```

`--graph`: Shows the commit graph.
`--all`: Includes all branches instead of just the current.

Inspecting the commit graph

Switching to another branch: `git checkout`

```
$ git checkout interface
Switched to branch 'interface'
```

This points **`HEAD`** to the named branch and updates your working directory.

Inspecting the commit graph

Switching to another branch: `git checkout`

```
$ git checkout interface
Switched to branch 'interface'
```

This points **HEAD** to the named branch and updates your working directory.

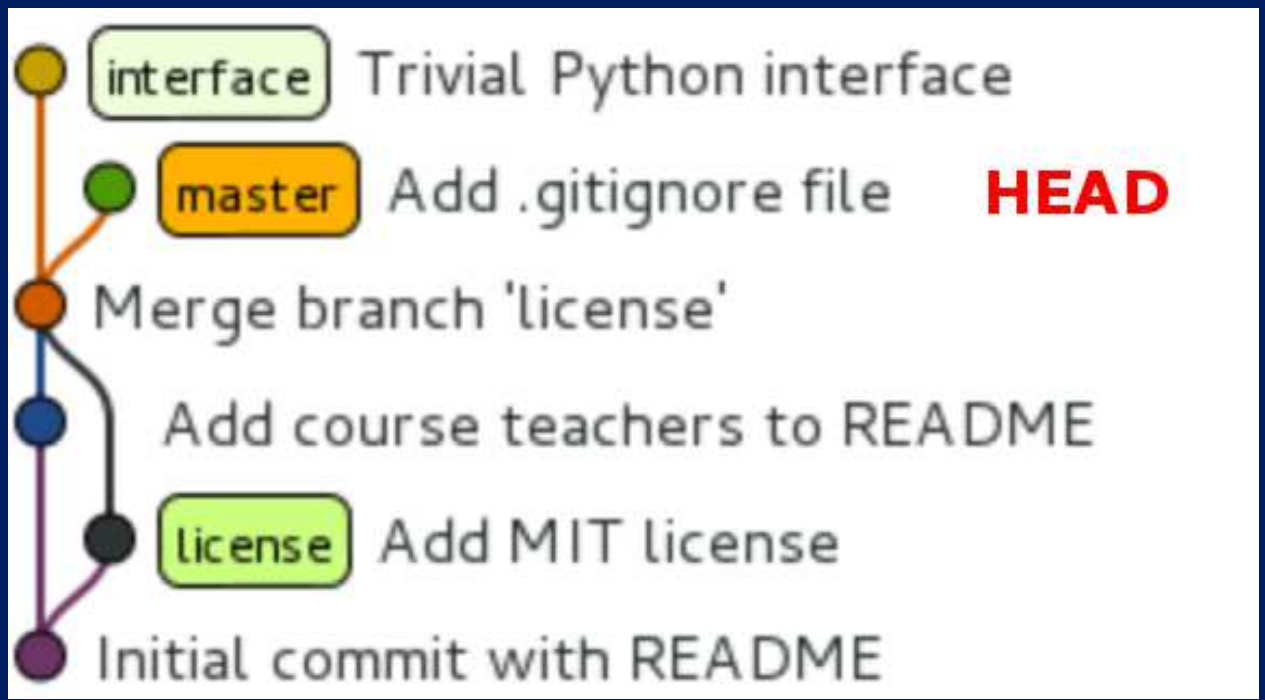
```
$ git status
# On branch interface
nothing to commit (working directory clean)
```

Manipulating the commit graph

Table of contents

Manipulating the commit graph

An example repository



- Branches **`license`** and **`interface`** diverged from **`master`**.
- Only **`license`** has been merged back into **`master`**.
- Current branch is **`master`**.

Manipulating the commit graph

Creating a new branch: `git branch`

```
$ git branch lib
```

Creates branch **`lib`** at **`HEAD`**.

Manipulating the commit graph

Creating a new branch: `git branch`

```
$ git branch lib
```

Creates branch **`lib`** at **`HEAD`**.

```
$ git checkout lib
Switched to branch 'lib'
```

Manipulating the commit graph

Creating a new branch: `git branch`

```
$ git branch lib
```

Creates branch **`lib`** at **`HEAD`**.

```
$ git checkout lib
Switched to branch 'lib'
```

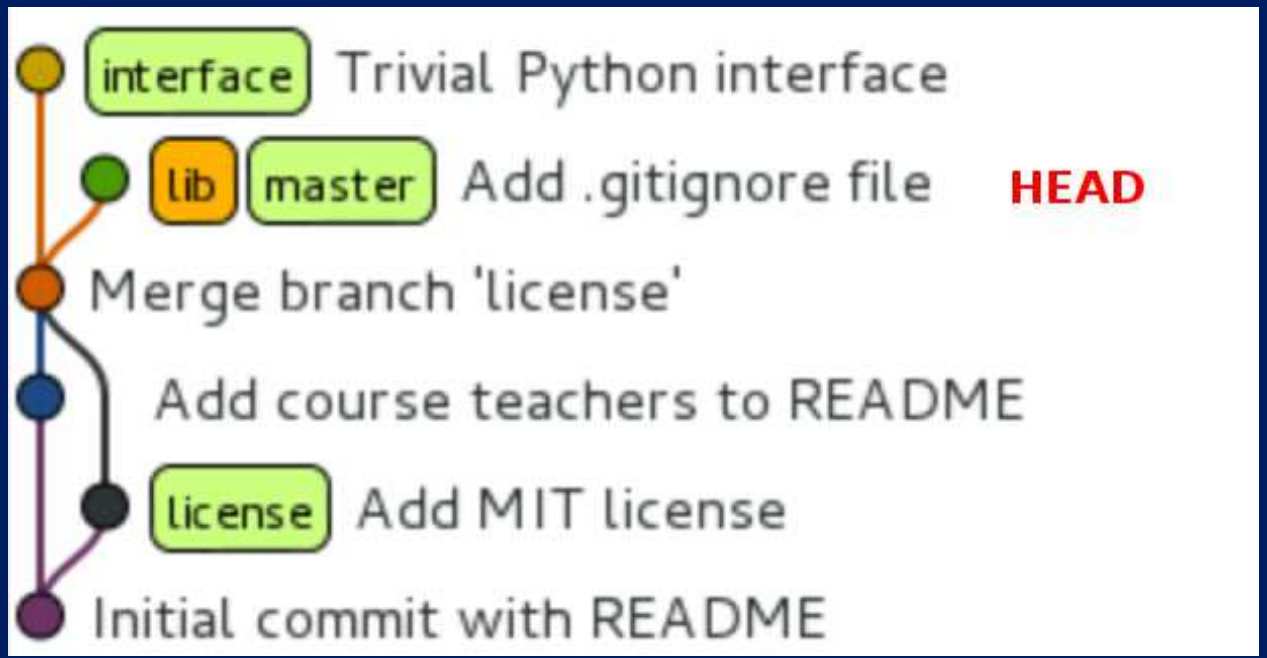
Or, for Git pro's, in one command:

```
$ git checkout -b lib
Switched to a new branch 'lib'
```

At this point, **`lib`** is just a label to the same commit as **`master`**.

Manipulating the commit graph

Our example repository



Created branch `lib` from `master`.

Manipulating the commit graph

Working on a branch: `git commit`

We can extend the **`lib`** branch by adding commits.

```
$ echo 'VERSION=1' > testlib.py
$ git add testlib.py
$ git commit -m 'Add empty testlib'
```

Manipulating the commit graph

Working on a branch: `git commit`

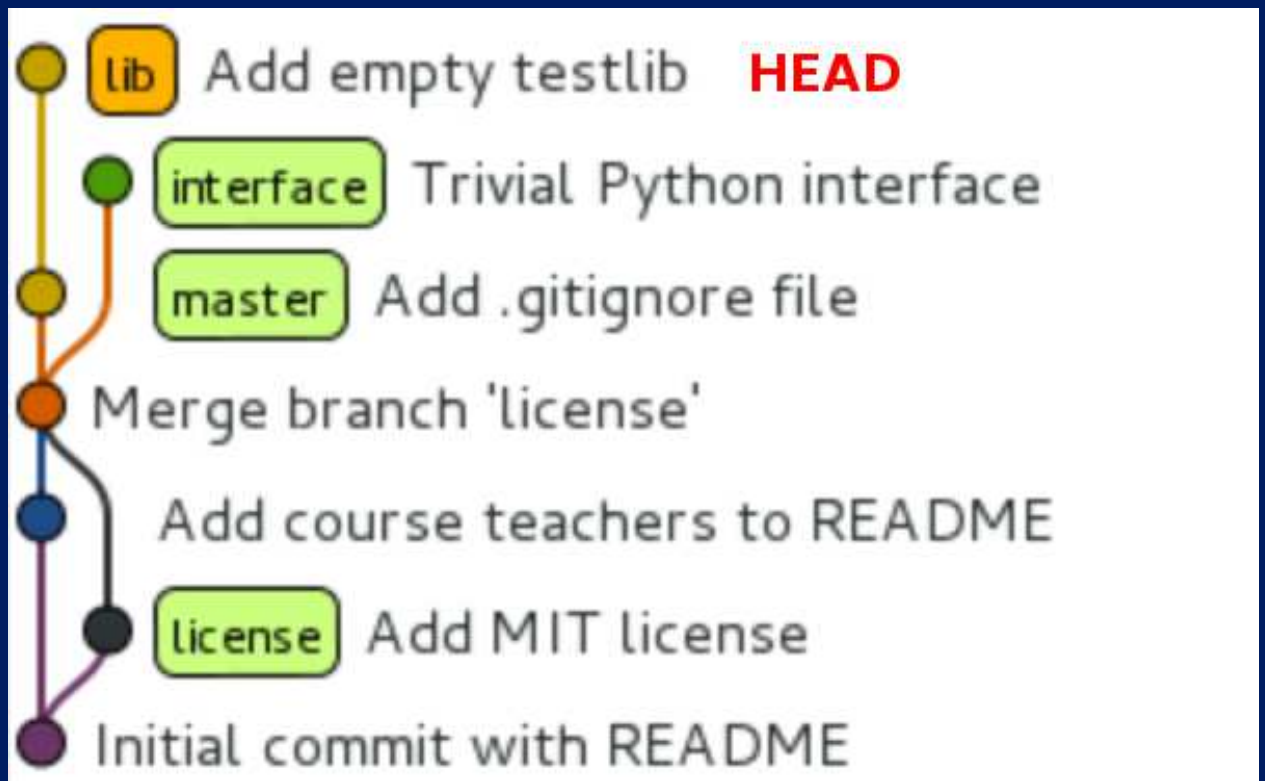
We can extend the **`lib`** branch by adding commits.

```
$ echo 'VERSION=1' > testlib.py
$ git add testlib.py
$ git commit -m 'Add empty testlib'

$ git log --oneline --decorate --graph --all
* 251a51b (HEAD, lib) Add empty testlib
* c7f3bd9 (master) Add .gitignore file
| * 8fc25c1 (interface) Trivial Python interface
|/
*   4a44c4e Merge branch 'license'
|\
| * 0fbe3e3 (license) Add MIT license
...
```

Manipulating the commit graph

Our example repository



Extended branch `lib`.

Manipulating the commit graph

Fast forward merging: `git merge`

Let's merge our work on **`lib`** back into **`master`**.

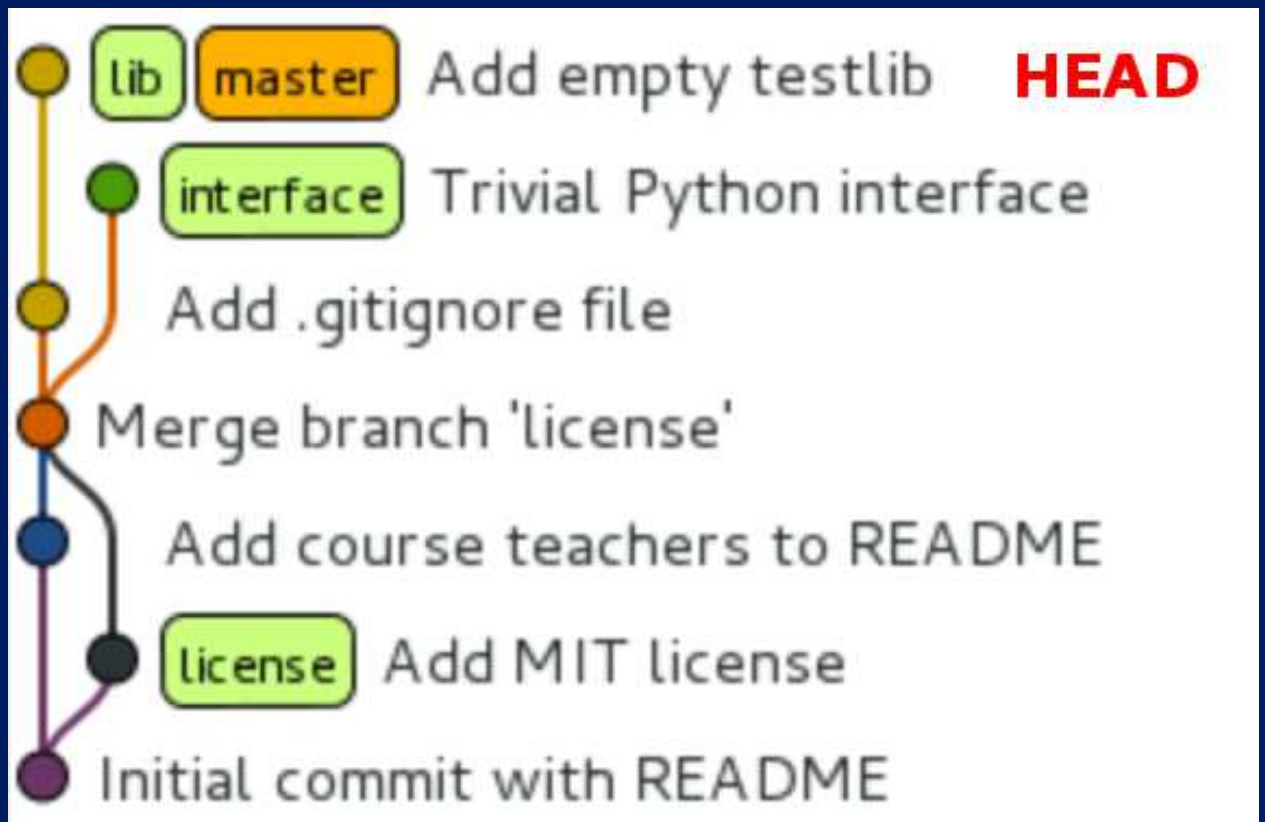
```
$ git checkout master
Switched to branch 'master'
$ git merge lib
Updating c7f3bd9..251a51b
Fast-forward
 testlib.py |    1 +
 1 file changed, 1 insertion(+)
 create mode 100644 testlib.py
```

This merge was easy:

- **`lib`** was directly upstream of **`master`**.
- Git just moves **`HEAD`** and **`master`** to point to **`lib`**.
- This is called a *fast forward merge*.

Manipulating the commit graph

Our example repository



Merged branch `lib` into `master` (fast forward).

Manipulating the commit graph

Three-way merging: `git merge`

Let's merge our work on **`interface`** back into **`master`**.

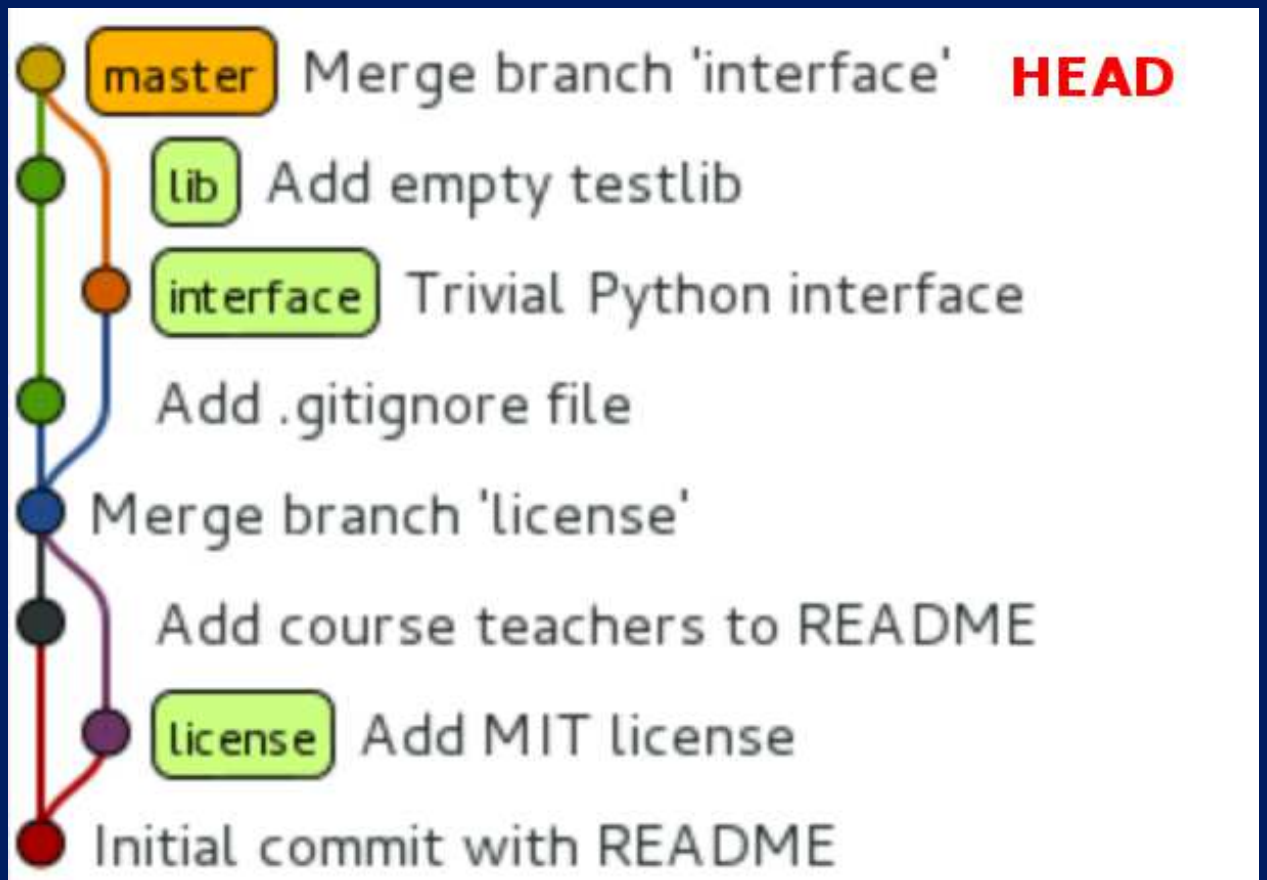
```
$ git merge interface
Merge made by the 'recursive' strategy.
 interface.py |    3 +++
 1 file changed, 3 insertions(+)
 create mode 100755 interface.py
```

This merge was harder:

- **`interface`** and **`master`** had diverged.
- Git determines changes in **`interface`** and **`master`** since their most recent common ancestor and creates a new commit from that.
- This is called a *recursive three-way merge* and the resulting merge a *merge commit*.

Manipulating the commit graph

Our example repository



Merged branch **`interface`** into **`master`**.

Manipulating the commit graph

Deleting branches: `git branch`

Old branches that have been merged can be deleted.

```
$ git branch -d interface
Deleted branch interface (was 8fc25c1).
```

```
$ git branch -d lib license
Deleted branch lib (was 251a51b).
Deleted branch license (was 0fbe3e3).
```

No history is lost, just labels removed.

Basic merge conflicts

Table of contents

Basic merge conflicts

Merge conflicts

Git is pretty good at merging:

- The changes might have been in different files.
- Or in different parts of the same file.
- Git tries to figure out a sensible result.

Basic merge conflicts

Merge conflicts

Git is pretty good at merging:

- The changes might have been in different files.
- Or in different parts of the same file.
- Git tries to figure out a sensible result.

Sometimes this is not possible:

- The changes might be incompatible.
- When we try `git merge`, Git gives up.
- This situation is called a *merge conflict*.

Basic merge conflicts

Setting the stage (1/3)

```
$ git log --oneline --decorate --graph --all
* 9d2ad27 (HEAD, master) State character preference
| * 374ab60 (simpsons) State character preference
|/
* 4012f4f Initial commit
```

We'd like to merge branch **`simpsons`** into **`master`**.

Basic merge conflicts

Setting the stage (2/3)

The last commit on **`master`**:

```
$ git show --oneline
9d2ad27 State character preference
diff --git a/README b/README
index de15194..ef40359 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 Facts about television series
 ==============================
+
+My favorite character is Eric Cartman.
```

Basic merge conflicts

Setting the stage (3/3)

The last commit on **`simpsons`**:

```
$ git show --oneline simpsons
374ab60 State character preference
diff --git a/README b/README
index de15194..9dd729b 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 Facts about television series
 ==============================
+
+My favorite character is Homer Simpson.
```

Basic merge conflicts

Creating a merge conflict

```
$ git merge simpsons
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then
commit the result.


$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>.." as appropriate to
#    mark resolution)
#
#               both modified:      README
#
no changes added to commit (use "git add" and/or
"git commit -a")
```

Basic merge conflicts

Resolving a merge conflict (1/2)

```
$ cat README
Facts about television series
==============================

<<<<<<< HEAD
My favorite character is Eric Cartman.
=======
My favorite character is Homer Simpson.
>>>>>>> simpsons
```

What we had is under **`HEAD`**, what **`simpsons`** had is above **`simpsons`**.

Basic merge conflicts

Resolving a merge conflict (1/2)

```
$ cat README
Facts about television series
==============================

<<<<<<< HEAD
My favorite character is Eric Cartman.
=======
My favorite character is Homer Simpson.
>>>>>>> simpsons
```

What we had is under **`HEAD`**, what **`simpsons`** had is above **`simpsons`**.

```
$ emacs README
```

Basic merge conflicts

Resolving a merge conflict (1/2)

We resolve the conflict by hand.

```
$ cat README
Facts about television series
==============================

My favorite characters are Eric Cartman
and Homer Simpson.
```

Basic merge conflicts

Resolving a merge conflict (1/2)

We resolve the conflict by hand.

```
$ cat README
Facts about television series
==============================

My favorite characters are Eric Cartman
and Homer Simpson.
```

And can now finish the merge commit.

```
$ git add README
$ git commit
[master 1e496cb] Merge branch 'simpsons'
```

Basic merge conflicts

Resolving conflicts with `git mergetool`

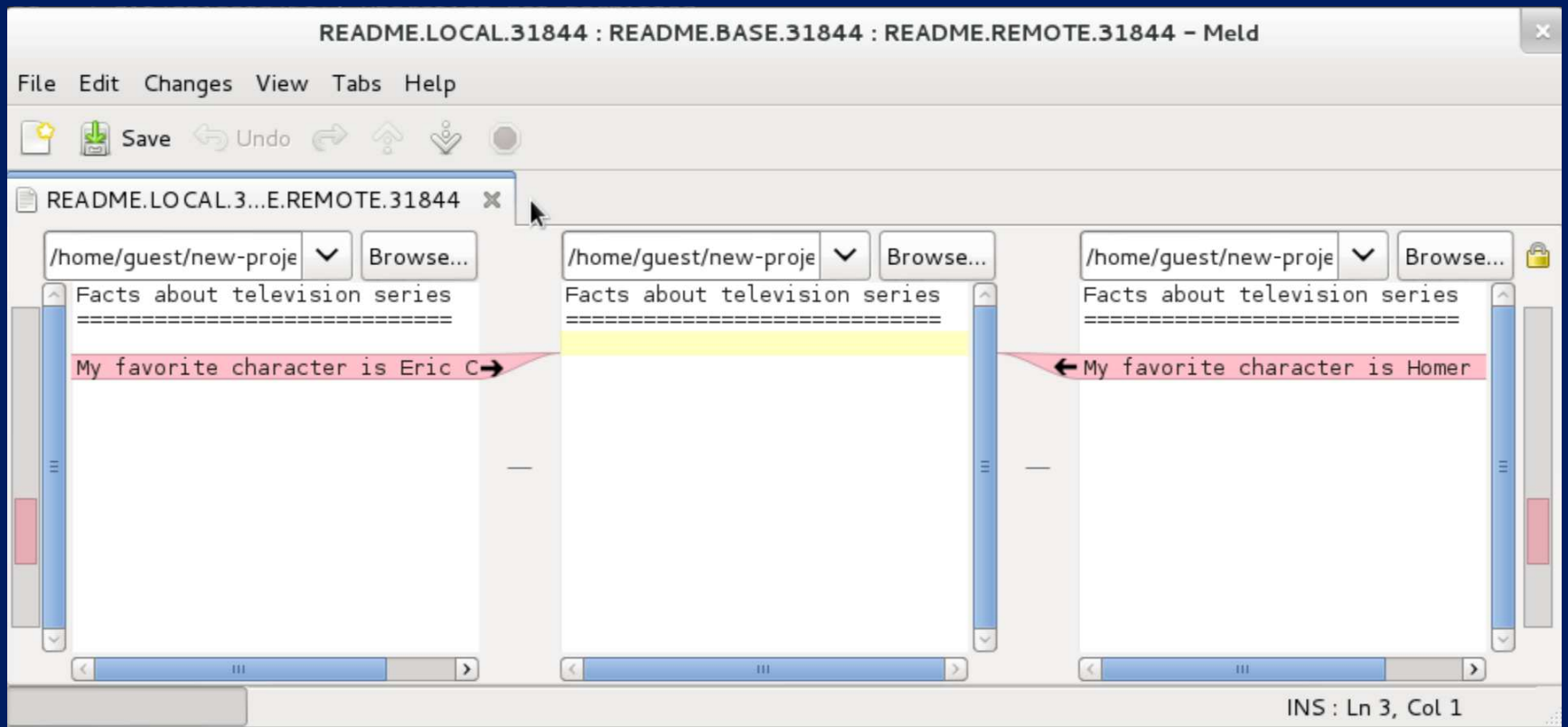
We can also use graphical merge tools such as *Meld*.

```
$ git mergetool
merge tool candidates: meld opendiff kdiff3 ...
Merging:
README

Normal merge conflict for 'README':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

Basic merge conflicts

Meld example



Tools like Meld provide an editable three-way diff.

Questions?

Table of contents

Questions?

Acknowledgements:

Jeroen Laros
Zuotian Tatum

http://git-scm.com/book

https://www.atlassian.com/git