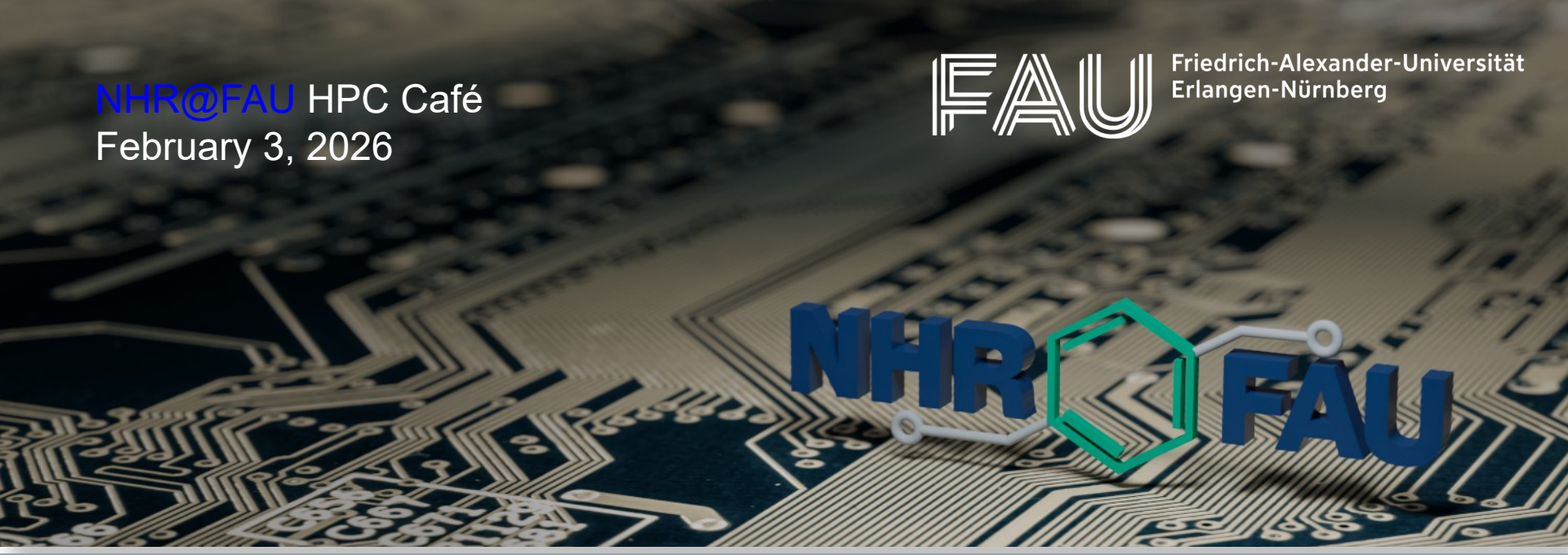


NHR@FAU HPC Café
February 3, 2026

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Git advanced usage

Michael Panzlaff

Erlangen National High Performance Computing Center (NHR@FAU)

Git advanced usage

Michael Panzlaff

Erlangen National High Performance Computing Center (NHR@FAU)

HPC Café, February 3, 2026

UPCOMING COURSES 2026

Erlangen National High Performance Computing Center (NHR@FAU)

Experience Parallel Programming ...

Feb 10-12	... with MPI+X	<<<hybrid>>>
Feb 24-26	... for HPC Systems	<<<on-site>>>
May 4-6	... with OpenMP	<<<online>>>
May 7-8	... with MPI	

Level up your GPU Programming ...

Mar 4-5	... with 10+ approaches	<<<online>>>
Mar 9	... with CUDA C/C++	
Mar 10-11	... for multiple GPUs	
Mar 27	... with CUDA Python	
Apr 8-10	... with modern C++	

Understand your Code's Performance ...

Apr 22-24	... on GPUs	<<<online>>>
Jun 9-12	... at Node-Level	

Learn about Software Development ...

Mar 12-13	... with Git	<<<online>>>

And did we mention AI?

Mar 25	... Deep Learning	<<<online>>>

Ready to level up?

Register at go-nhr.de/trainings

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

NHR FAU

NHR FAU

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Introduction

Introduction

So you probably know those commands?

- `git clone https://github.com/RRZE-HPC/likwid`
- `git pull`
- `git add main.c`
- `git commit -m “very much interesting!”`
- `git push`

Introduction

So you probably know those commands?

- `git clone https://github.com/RRZE-HPC/likwid`
- `git pull`
- `git add main.c`
- `git commit -m “very much interesting!”`
- `git push`

But do you know about?

- staging area

Introduction

So you probably know those commands?

- `git clone https://github.com/RRZE-HPC/likwid`
- `git pull`
- `git add main.c`
- `git commit -m “very much interesting!”`
- `git push`

But do you know about?

- staging area
- history

Introduction

So you probably know those commands?

- `git clone https://github.com/RRZE-HPC/likwid`
- `git pull`
- `git add main.c`
- `git commit -m “very much interesting!”`
- `git push`

But do you know about?

- staging area
- history
- reflog

Introduction

So you probably know those commands?

- `git clone https://github.com/RRZE-HPC/likwid`
- `git pull`
- `git add main.c`
- `git commit -m “very much interesting!”`
- `git push`

But do you know about?

- staging area
- history
- reflog

Don’t worry, this won’t be a “just theory” lesson :-)

Introduction

Have you seen those “errors” before?

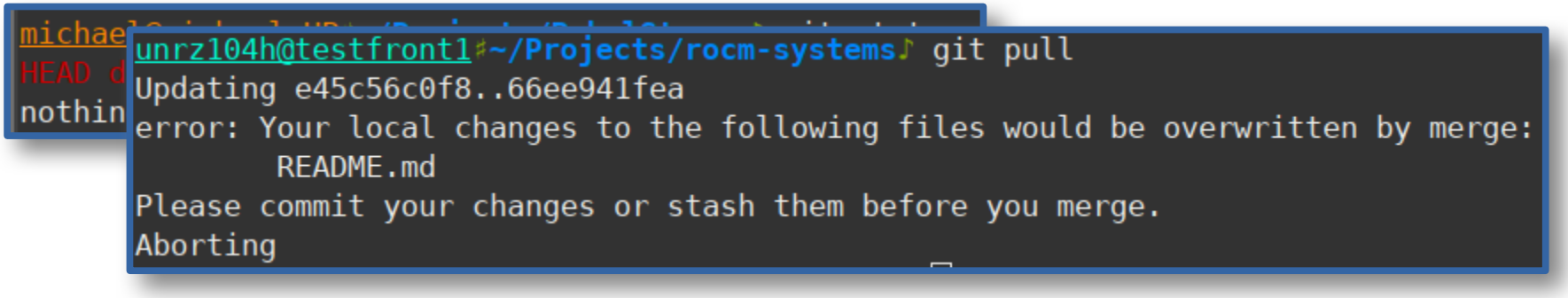
Introduction

Have you seen those “errors” before?

```
michael@michael-HP♯~/Projects/BabelStream♪ git status
HEAD detached at 78ba4ff
nothing to commit, working tree clean
```

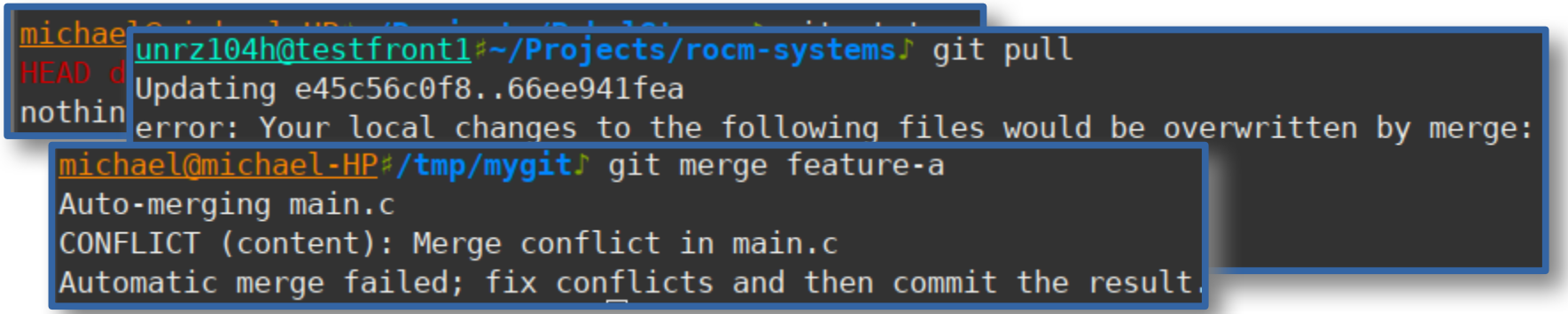
Introduction

Have you seen those “errors” before?



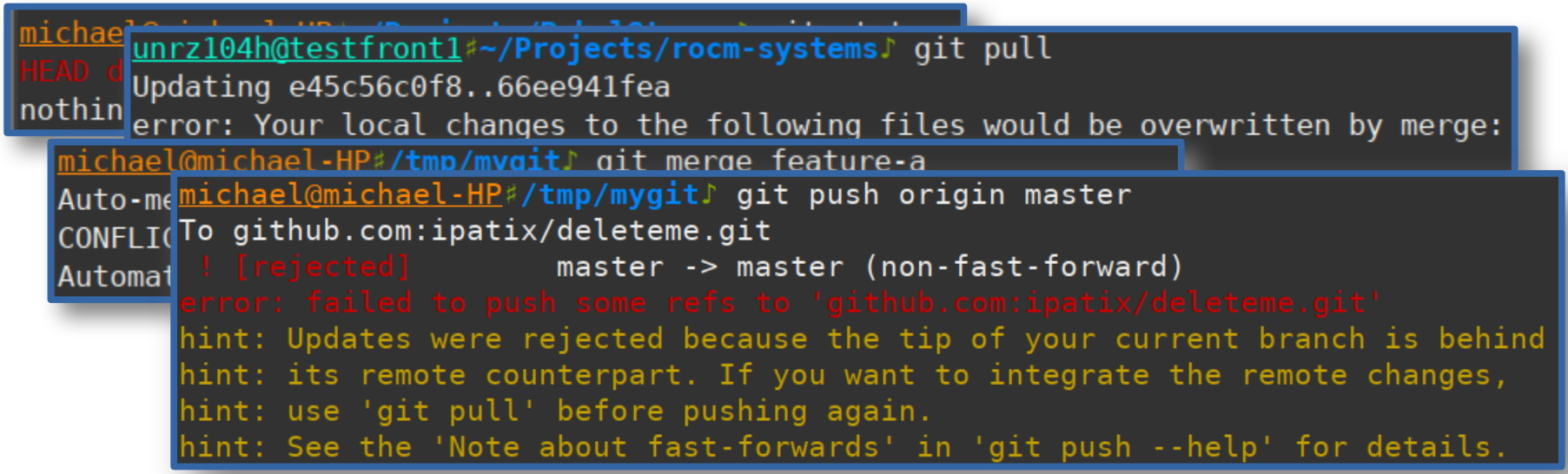
Introduction

Have you seen those “errors” before?



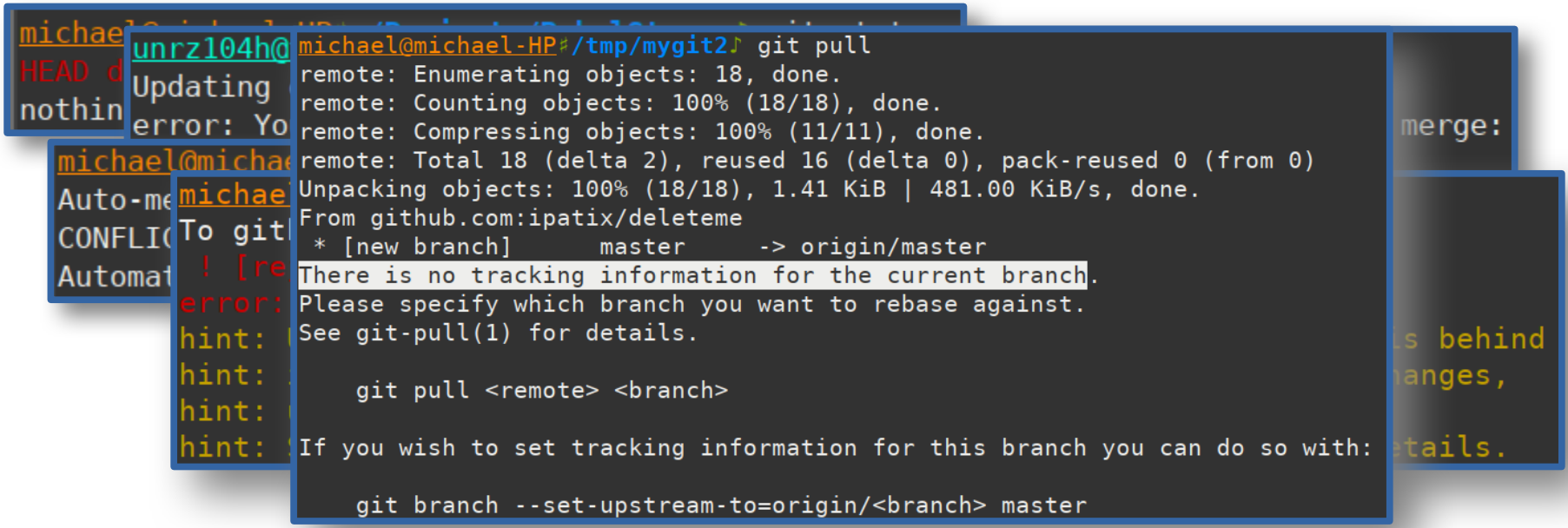
Introduction

Have you seen those “errors” before?



Introduction

Have you seen those “errors” before?



Introduction

“solid fundamentals → advanced user”

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Git history

Git history

First things first:

- What it is not: The history of the tool Git itself

Git history

First things first:

- What it is not: The history of the tool Git itself
- What it is: The history of commits of a Git repository

Git history

First things first:

- What it is not: The history of the tool Git itself
- What it is: The history of commits of a Git repository

Why is history relevant?

- We use it to look back in time.

Git history

First things first:

- What it is not: The history of the tool Git itself
- What it is: The history of commits of a Git repository

Why is history relevant?

- We use it to look back in time.

  → It should be easy to look back in time

Git history

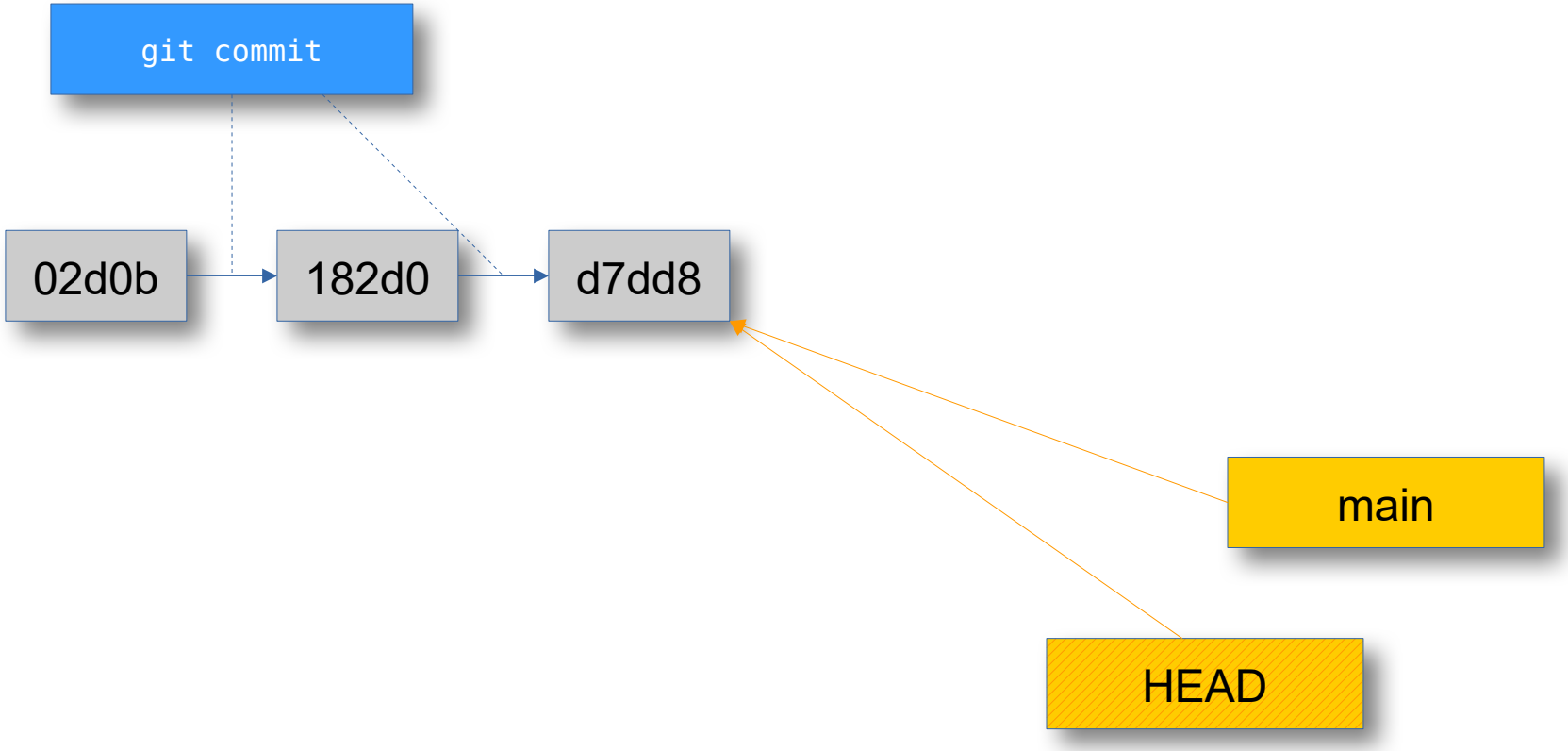
First things first:

- What it is not: The history of the tool Git itself
- What it is: The history of commits of a Git repository

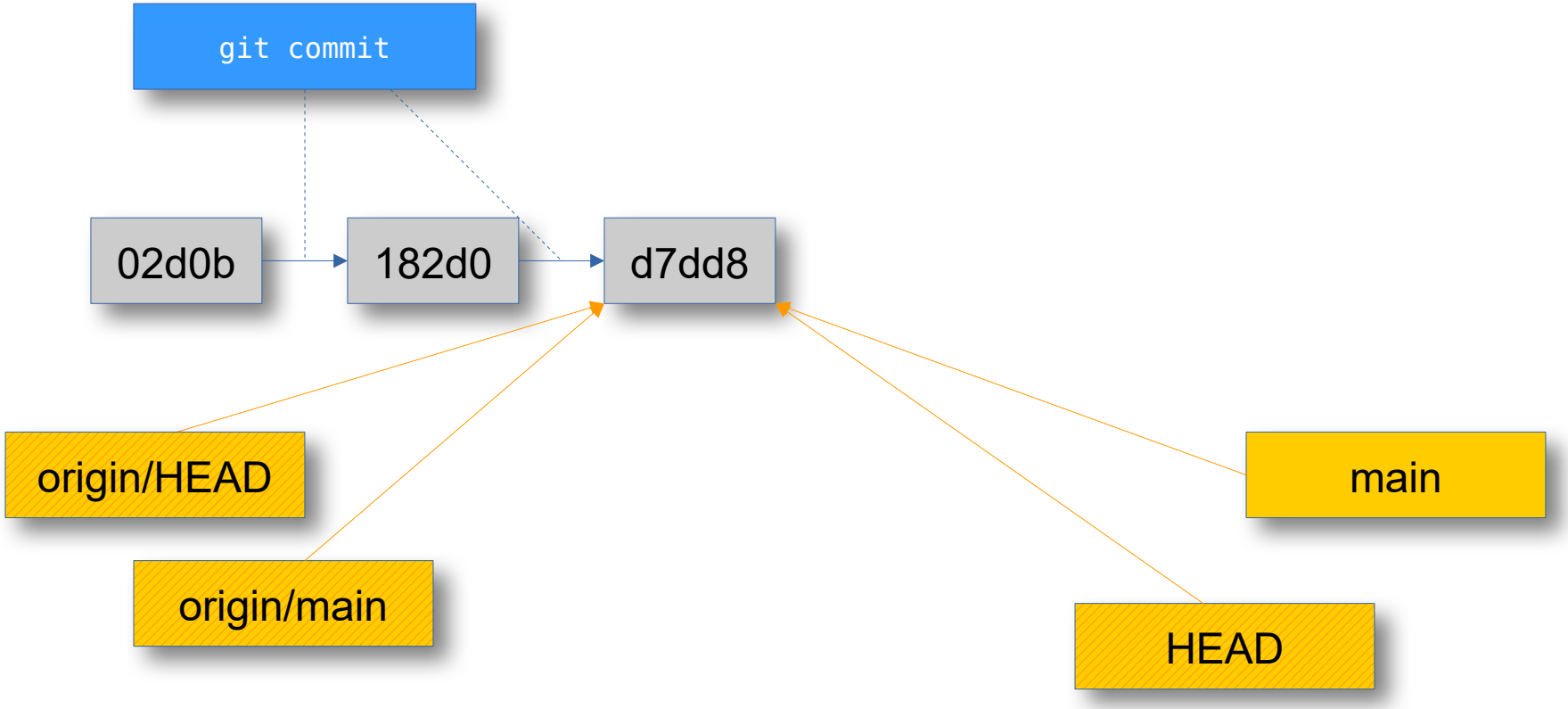
Why is history relevant?

- We use it to look back in time.
  → It should be easy to look back in time
- Because it is non-linear (directed acyclic graph)
  → complications during merge/rebase

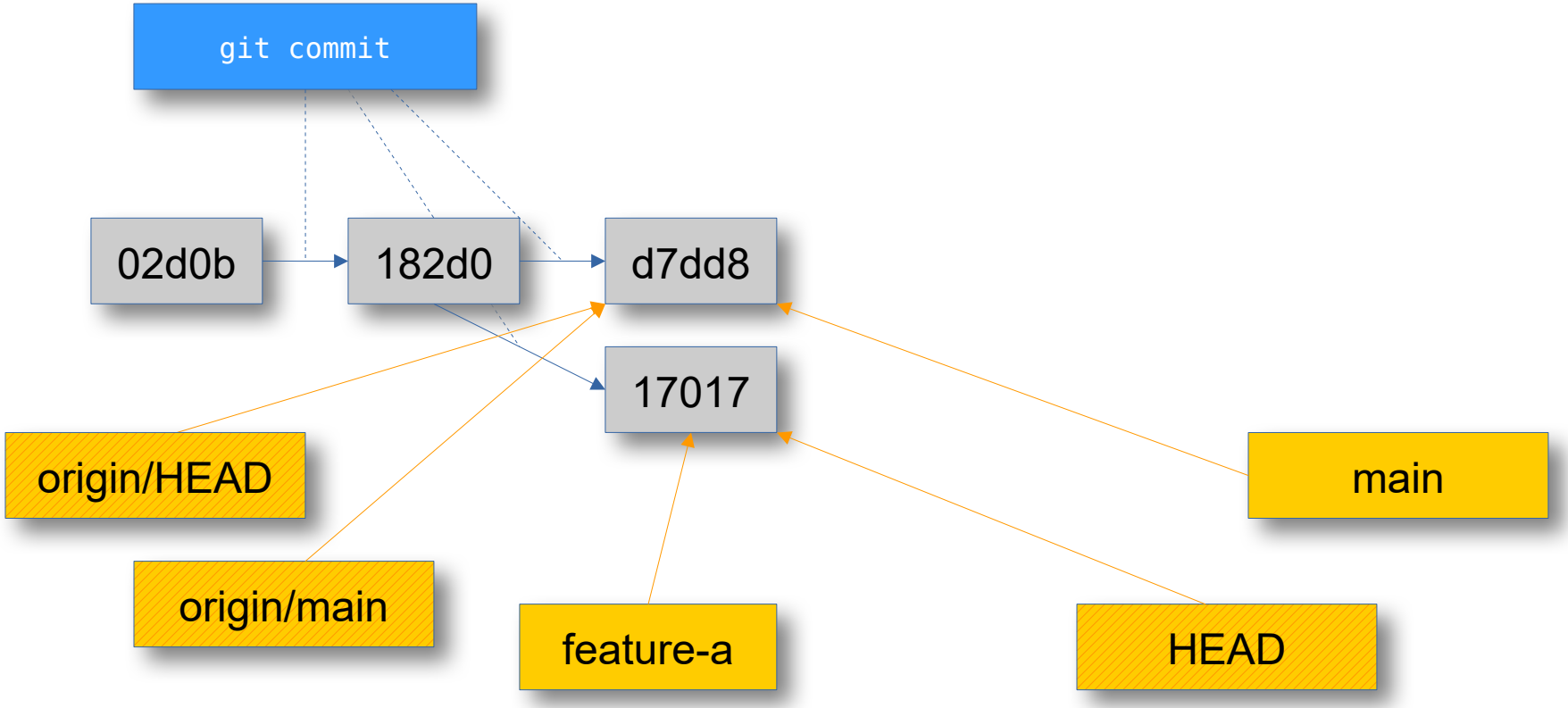
Git history (with merge)



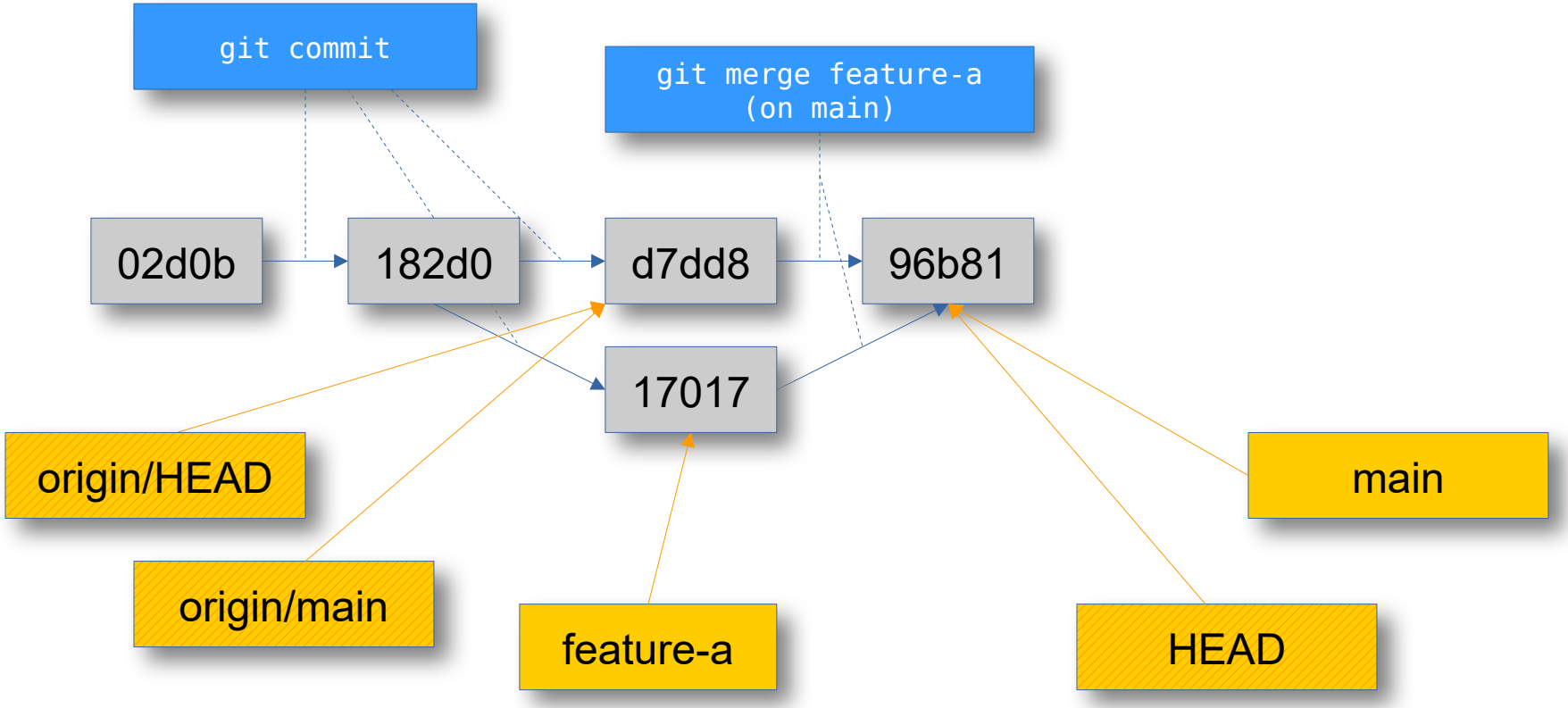
Git history (with merge)



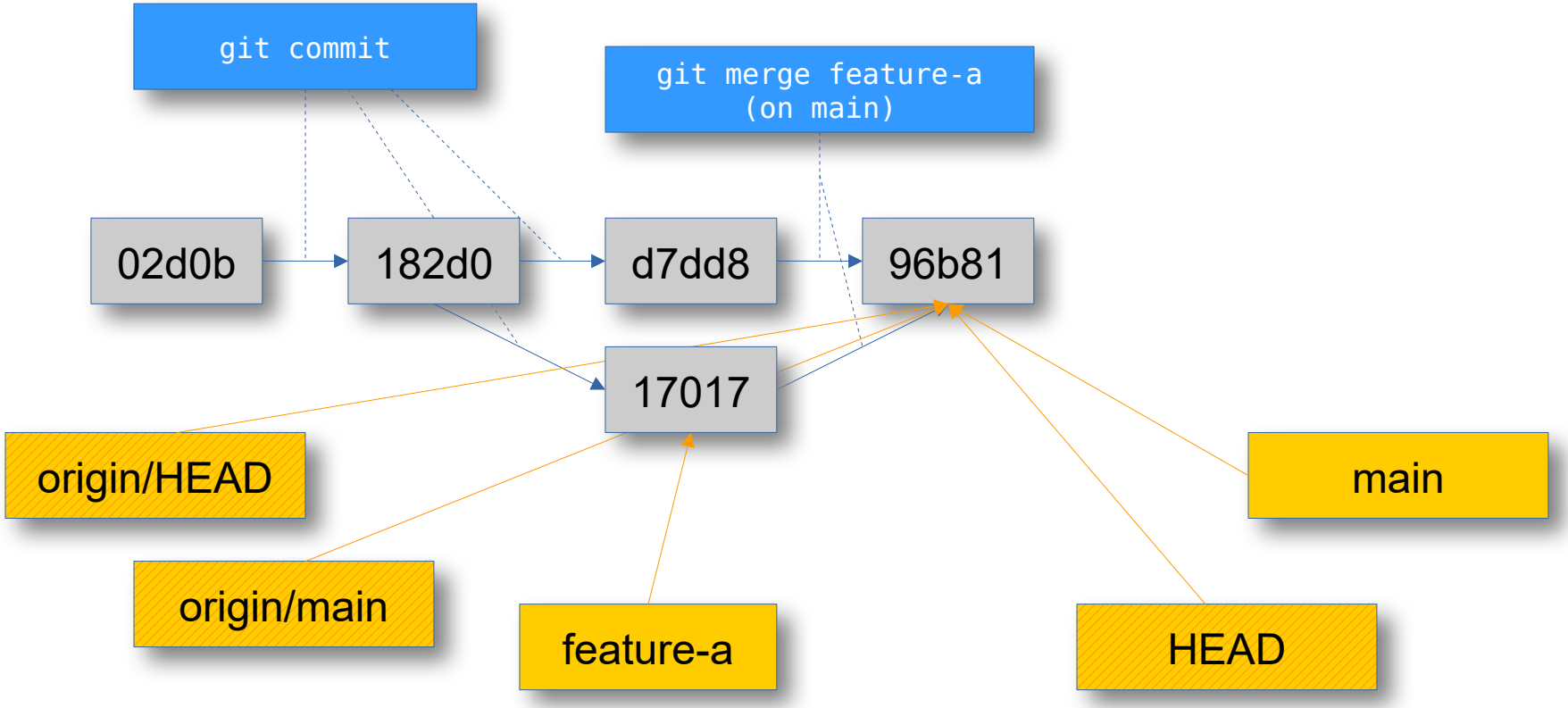
Git history (with merge)



Git history (with merge)



Git history (with merge)



Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` `# Create directory`

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` `# Create directory`
- `git init` `# Init Git repository`

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` `# Create directory`
- `git init` `# Init Git repository`
- `echo “hello” > file.txt` `# Create` **`file.txt`** `with some content`

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` `# Create directory`
- `git init` `# Init Git repository`
- `echo “hello” > file.txt` `# Create` **`file.txt`** `with some content`
- `git add file.txt && git commit` `# Create commit #1 (`**`02d0b`**`)`

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` `# Create directory`
- `git init` `# Init Git repository`
- `echo “hello” > file.txt` `# Create` **`file.txt`** `with some content`
- `git add file.txt && git commit` `# Create commit #1 (`**`02d0b`**`)`
- `echo “there” >> file.txt` `# Add some content to` **`file.txt`**

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` # Create directory
- `git init` # Init Git repository
- `echo “hello” > file.txt` # Create **file.txt** with some content
- `git add file.txt && git commit` # Create commit #1 (**02d0b**)
- `echo “there” >> file.txt` # Add some content to **file.txt**
- `git add file.txt && git commit` # Create commit #2 (**182d0**)

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` # Create directory
- `git init` # Init Git repository
- `echo “hello” > file.txt` # Create **file.txt** with some content
- `git add file.txt && git commit` # Create commit #1 (**02d0b**)
- `echo “there” >> file.txt` # Add some content to **file.txt**
- `git add file.txt && git commit` # Create commit #2 (**182d0**)
- `echo “NHR” >> file.txt` # Add more content to **file.txt**

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` # Create directory
- `git init` # Init Git repository
- `echo “hello” > file.txt` # Create **file.txt** with some content
- `git add file.txt && git commit` # Create commit #1 (**02d0b**)
- `echo “there” >> file.txt` # Add some content to **file.txt**
- `git add file.txt && git commit` # Create commit #2 (**182d0**)
- `echo “NHR” >> file.txt` # Add more content to **file.txt**
- `git add file.txt && git commit` # Create commit #3 (**d7dd8**)

Git history (with merge)

Reproduce step 1:

- `mkdir myrepo && cd myrepo` # Create directory
- `git init` # Init Git repository
- `echo “hello” > file.txt` # Create **file.txt** with some content
- `git add file.txt && git commit` # Create commit #1 (**02d0b**)
- `echo “there” >> file.txt` # Add some content to **file.txt**
- `git add file.txt && git commit` # Create commit #2 (**182d0**)
- `echo “NHR” >> file.txt` # Add more content to **file.txt**
- `git add file.txt && git commit` # Create commit #3 (**d7dd8**)

Commit hashes will differ for you!

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**
- `git push --set-upstream-to origin main`
  `# Push the current branch remote` **`origin`** `and mark it as upstream`

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**
- `git push --set-upstream-to origin main`
  `# Push the current branch remote` **`origin`** `and mark it as upstream`

Reproduce step 3:

- `git switch 182d0` `# Switch to a previous commit`

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**
- `git push --set-upstream-to origin main`
  `# Push the current branch remote` **`origin`** `and mark it as upstream`

Reproduce step 3:

- `git switch 182d0` `# Switch to a previous commit`
- `git switch --create feature-a` `# Create branch` **`feature-a`** `and switch to it`

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**
- `git push --set-upstream-to origin main`
  `# Push the current branch remote` **`origin`** `and mark it as upstream`

Reproduce step 3:

- `git switch 182d0` `# Switch to a previous commit`
- `git switch --create feature-a` `# Create branch` **`feature-a`** `and switch to it`
- `echo “FAU” > file2.txt` `# Create` **`file2.txt`**

Git history (with merge)

Reproduce step 2:

- `git remote add origin “git@github.com:myuser/myrepo.git”`
  `# Add a remote called` **`origin`**
- `git push --set-upstream-to origin main`
  `# Push the current branch remote` **`origin`** `and mark it as upstream`

Reproduce step 3:

- `git switch 182d0` `# Switch to a previous commit`
- `git switch --create feature-a` `# Create branch` **`feature-a`** `and switch to it`
- `echo “FAU” > file2.txt` `# Create` **`file2.txt`**
- `git add file2.txt && git commit` `# Create commit #4 (`**`17017`**`)`

Git history (with merge)

Reproduce step 4:

- `git switch main` `# Switch back to **main** branch`

Git history (with merge)

Reproduce step 4:

- `git switch main` `# Switch back to **main** branch`
- `git merge feature-a` `# Merge changes from **feature-a** into **main**`

Git history (with merge)

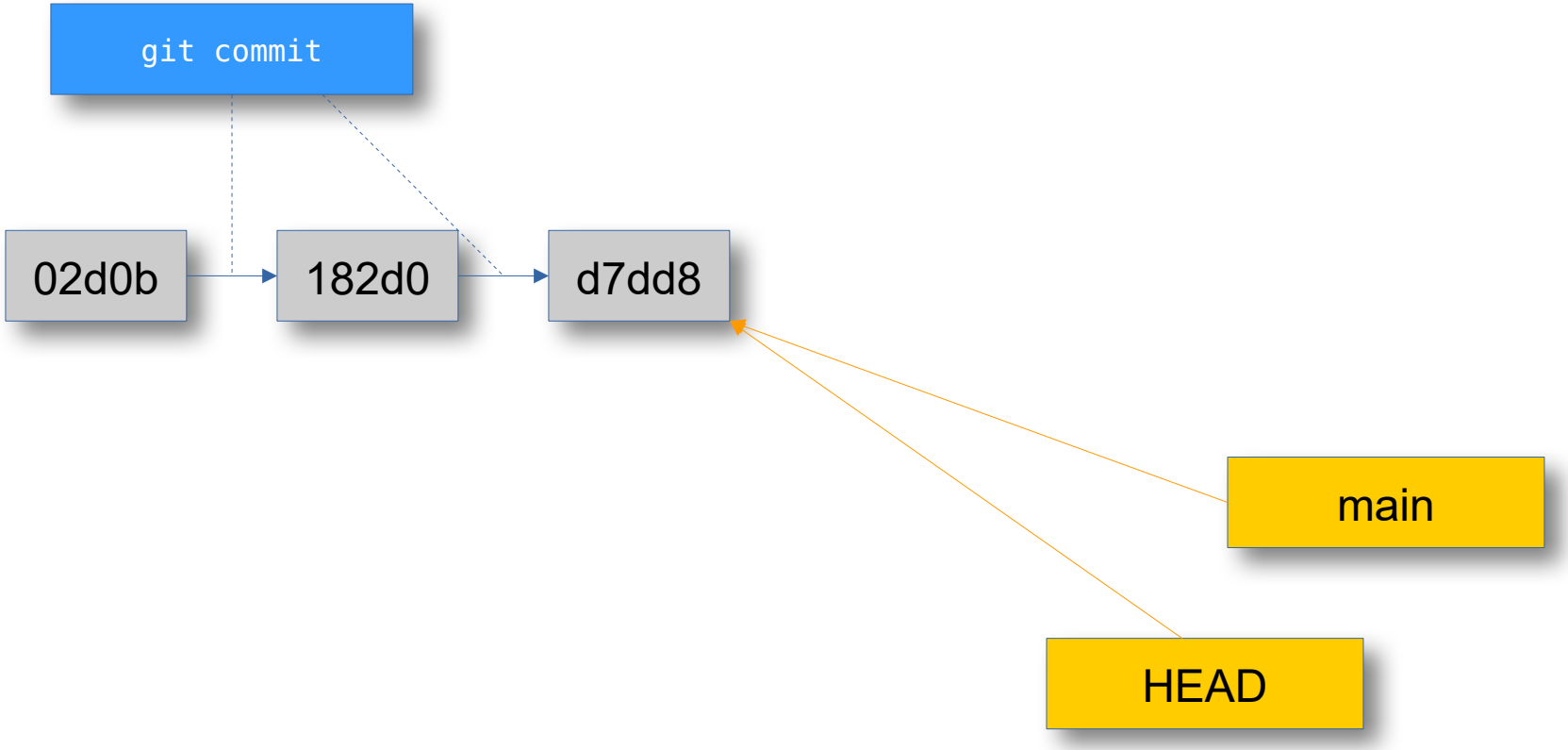
Reproduce step 4:

- `git switch main` `# Switch back to **main** branch`
- `git merge feature-a` `# Merge changes from **feature-a** into **main**`

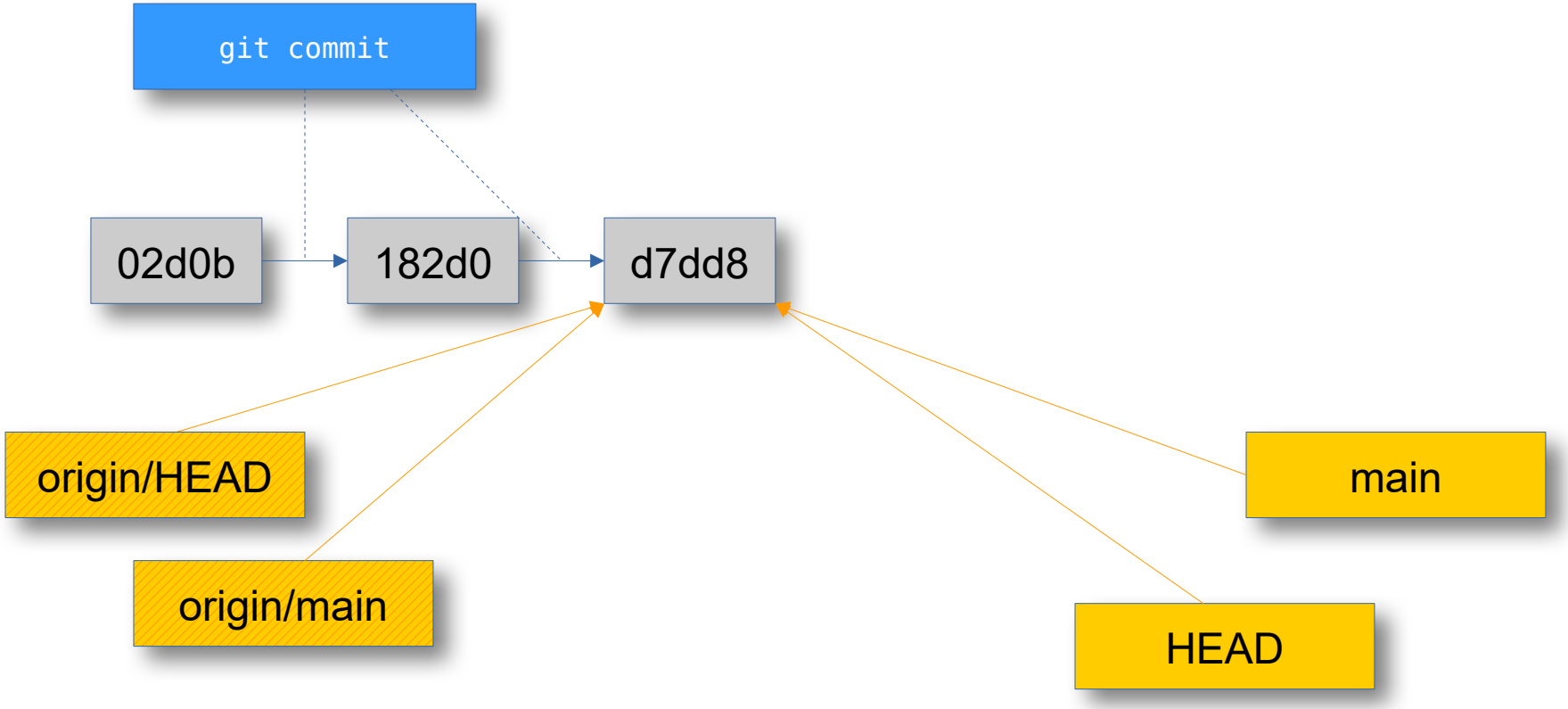
Reproduce step 5:

- `git push` `# Push new state of **main** branch to **origin**`

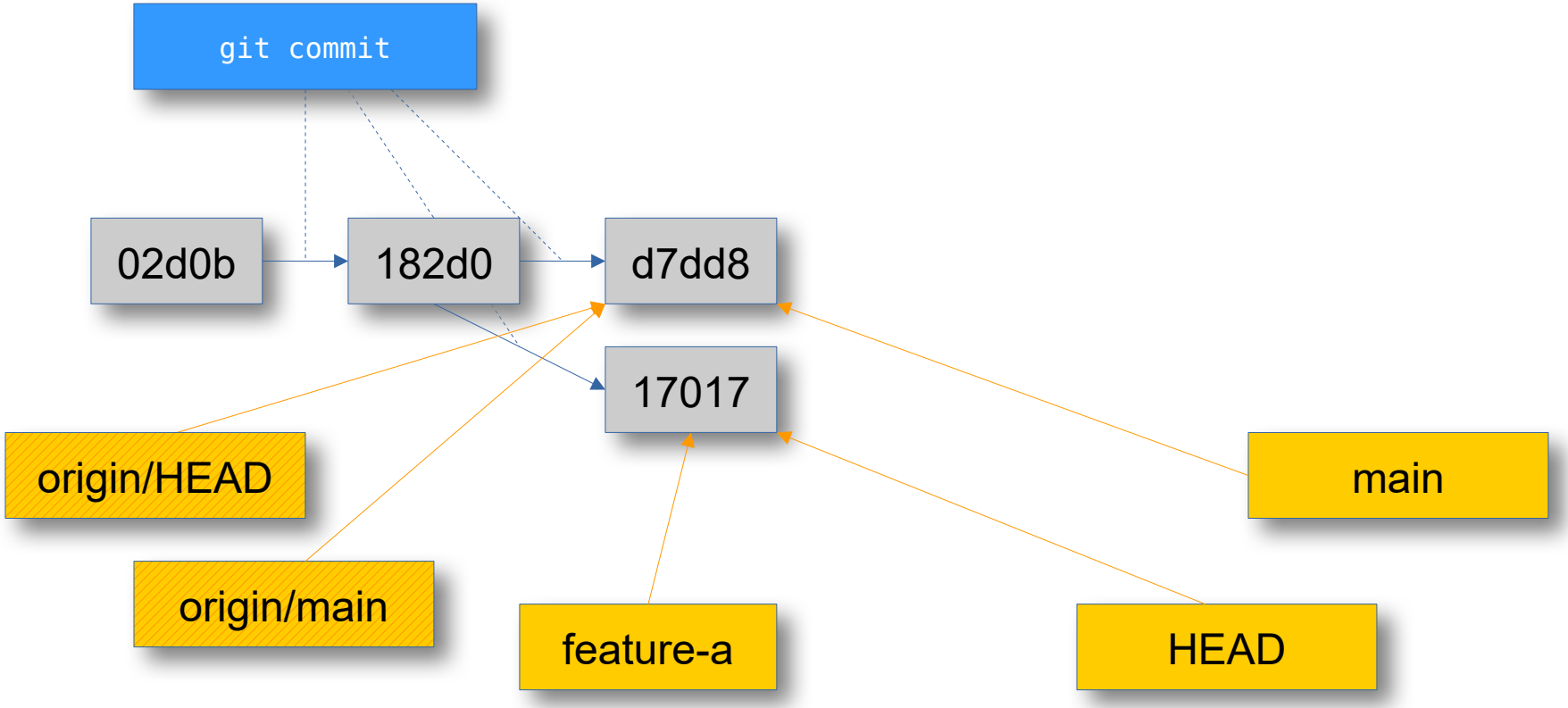
Git history (with rebase)



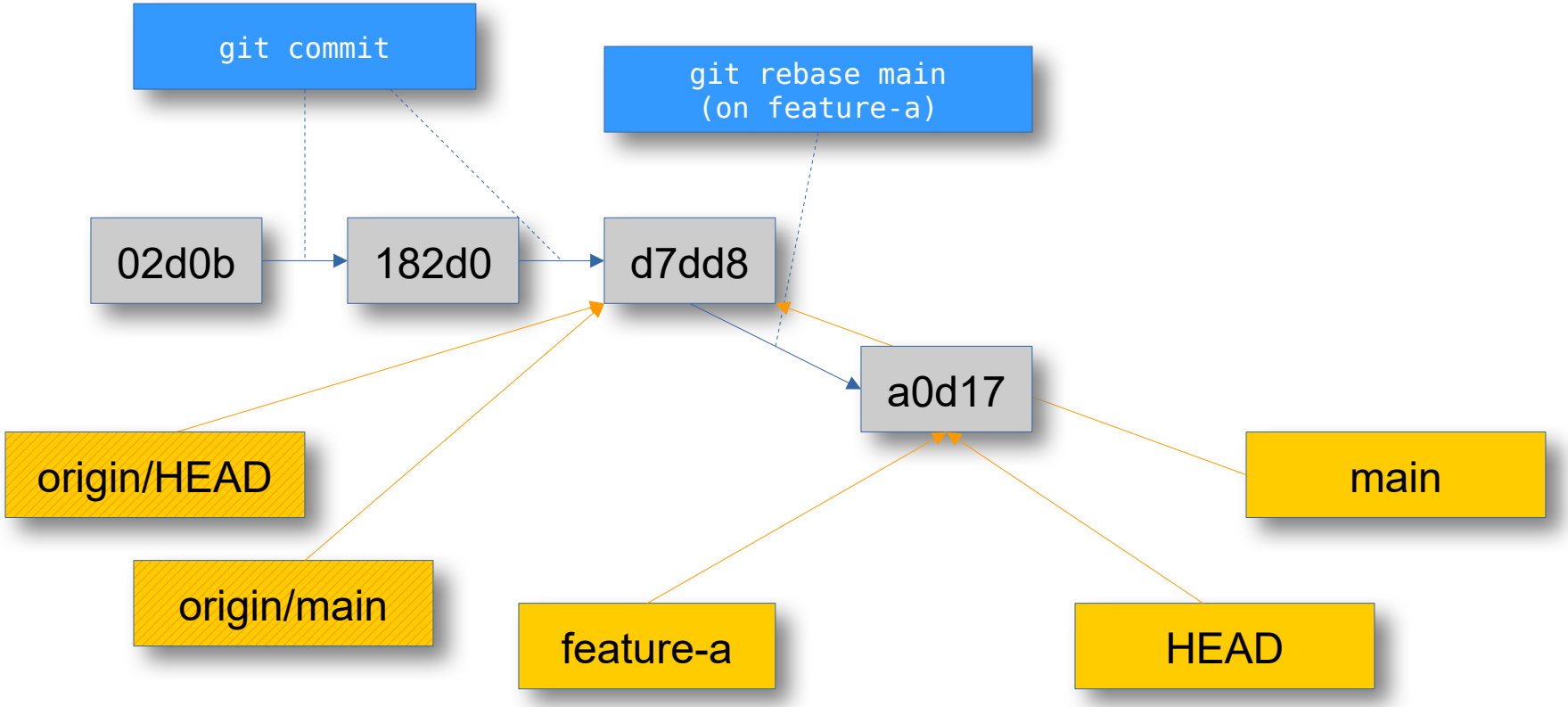
Git history (with rebase)



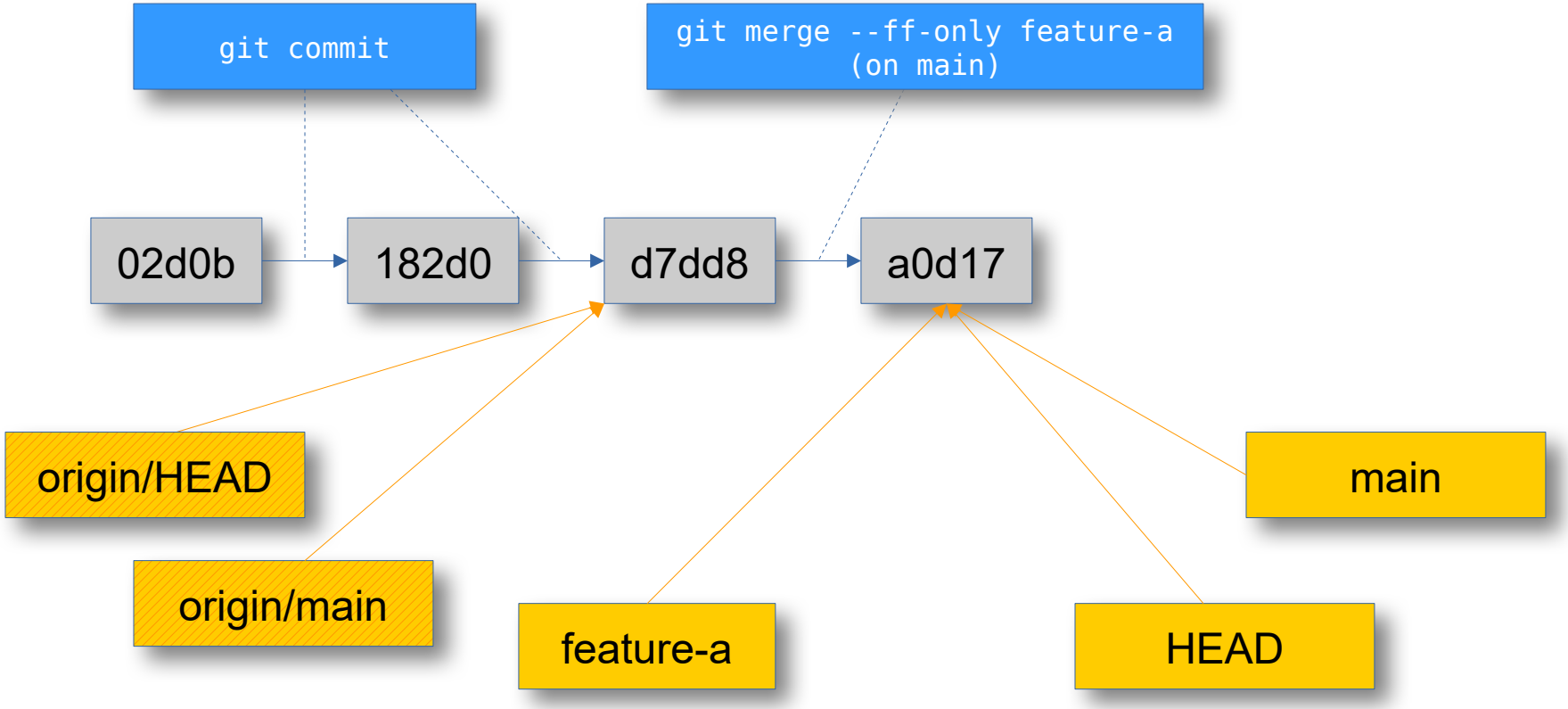
Git history (with rebase)



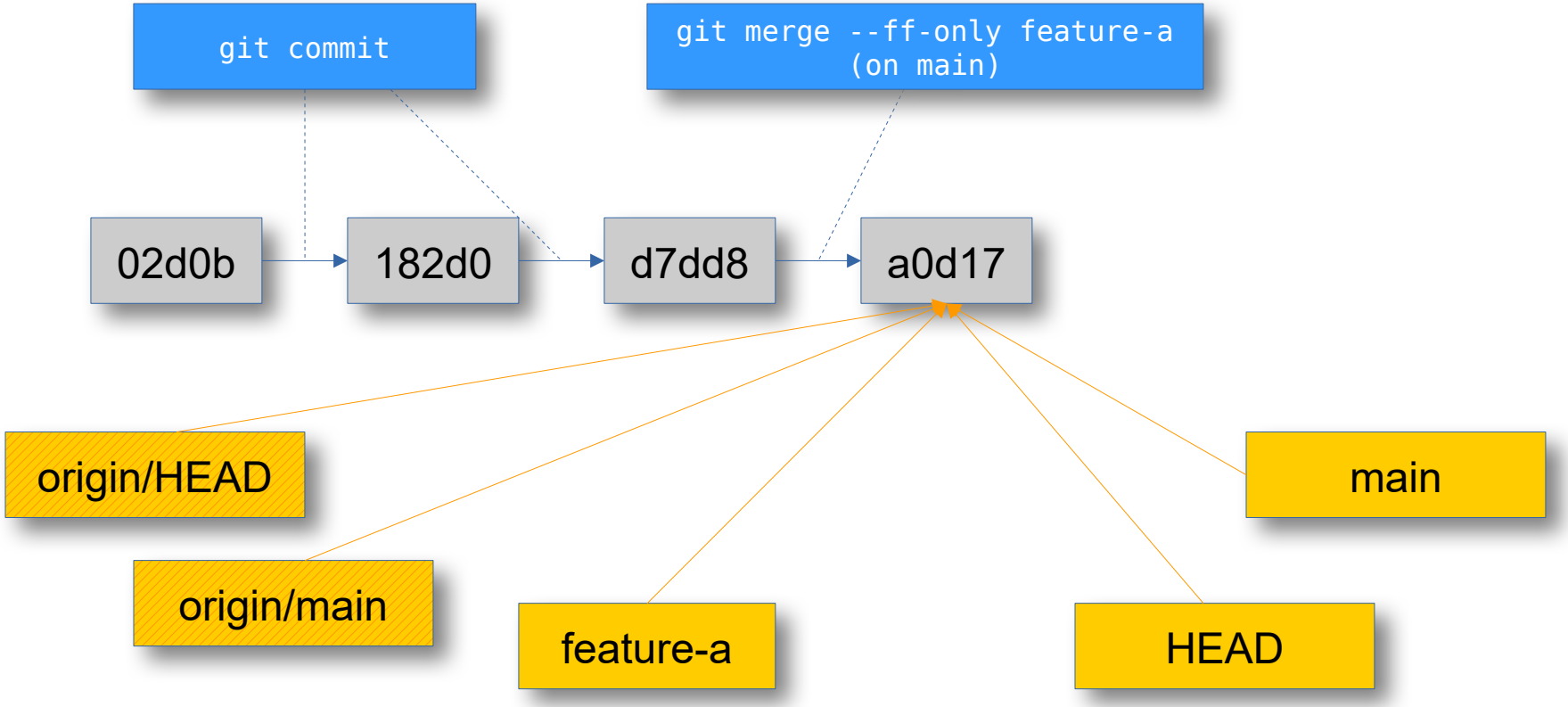
Git history (with rebase)



Git history (with rebase)



Git history (with rebase)



Git history (with rebase)

Reproduce step 4:

- `git rebase main` # Rewrite history so that **feature-a**'s commits are ontop of **main**

Git history (with rebase)

Reproduce step 4:

- `git rebase main` # Rewrite history so that **feature-a**'s commits are ontop of **main**

Reproduce step 5:

- `git switch main` # Switch to **main**

Git history (with rebase)

Reproduce step 4:

- `git rebase main` # Rewrite history so that **feature-a**'s commits are ontop of **main**

Reproduce step 5:

- `git switch main` # Switch to **main**
- `git merge --ff-only feature-a` # Merge **feature-a** into **main**, but do # not create an explicit merge commit

Git history (with rebase)

Reproduce step 4:

- `git rebase main` # Rewrite history so that **feature-a**'s commits are ontop of **main**

Reproduce step 5:

- `git switch main` # Switch to **main**
- `git merge --ff-only feature-a` # Merge **feature-a** into **main**, but do # not create an explicit merge commit

Reproduce step 6:

- same as "merge" example

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so
- Avoids future merge conflicts. Immediate resolution is required on rebase.

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so
- Avoids future merge conflicts. Immediate resolution is required on rebase.
- Avoids “opqaue” merge commits (“does this merge commit produce functional code?”)

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so
- Avoids future merge conflicts. Immediate resolution is required on rebase.
- Avoids “opqaue” merge commits (“does this merge commit produce functional code?”)
- → clean and linear “easy to read” history

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so
- Avoids future merge conflicts. Immediate resolution is required on rebase.
- Avoids “opqaue” merge commits (“does this merge commit produce functional code?”)
- → clean and linear “easy to read” history

Why not rebase?

- One conflict may have to be resolved more than once (→ more work).

Git history (merge vs. rebase)

Why rebase (at all)?

- Project maintainers may ask you to do so
- Avoids future merge conflicts. Immediate resolution is required on rebase.
- Avoids “opqaue” merge commits (“does this merge commit produce functional code?”)
- → clean and linear “easy to read” history

Why not rebase?

- One conflict may have to be resolved more than once (→ more work).
- Other people are working on your branch (→ do not delete someone’s base commit).

Git history (merge vs. rebase)

*Example: history comparison **cc-backend** and **likwid***

Git history (pull)

So what does `git pull` actually do?

Git history (pull)

So what does `git pull` actually do?

- `git fetch`
- `git merge` **`origin/mybranch`** (remote and branch determined via “tracking”)

Git history (pull)

So what does `git pull` actually do?

- `git fetch`
- `git merge` **`origin/mybranch`** (remote and branch determined via “tracking”)

It can also rebase (with `git config --global pull.rebase true`):

- `git fetch`
- `git rebase` **`origin/mybranch`**

Git history (pull)

So what does `git pull` actually do?

- `git fetch`
- `git merge` **`origin/mybranch`** (remote and branch determined via “tracking”)

It can also rebase (with `git config --global pull.rebase true`):

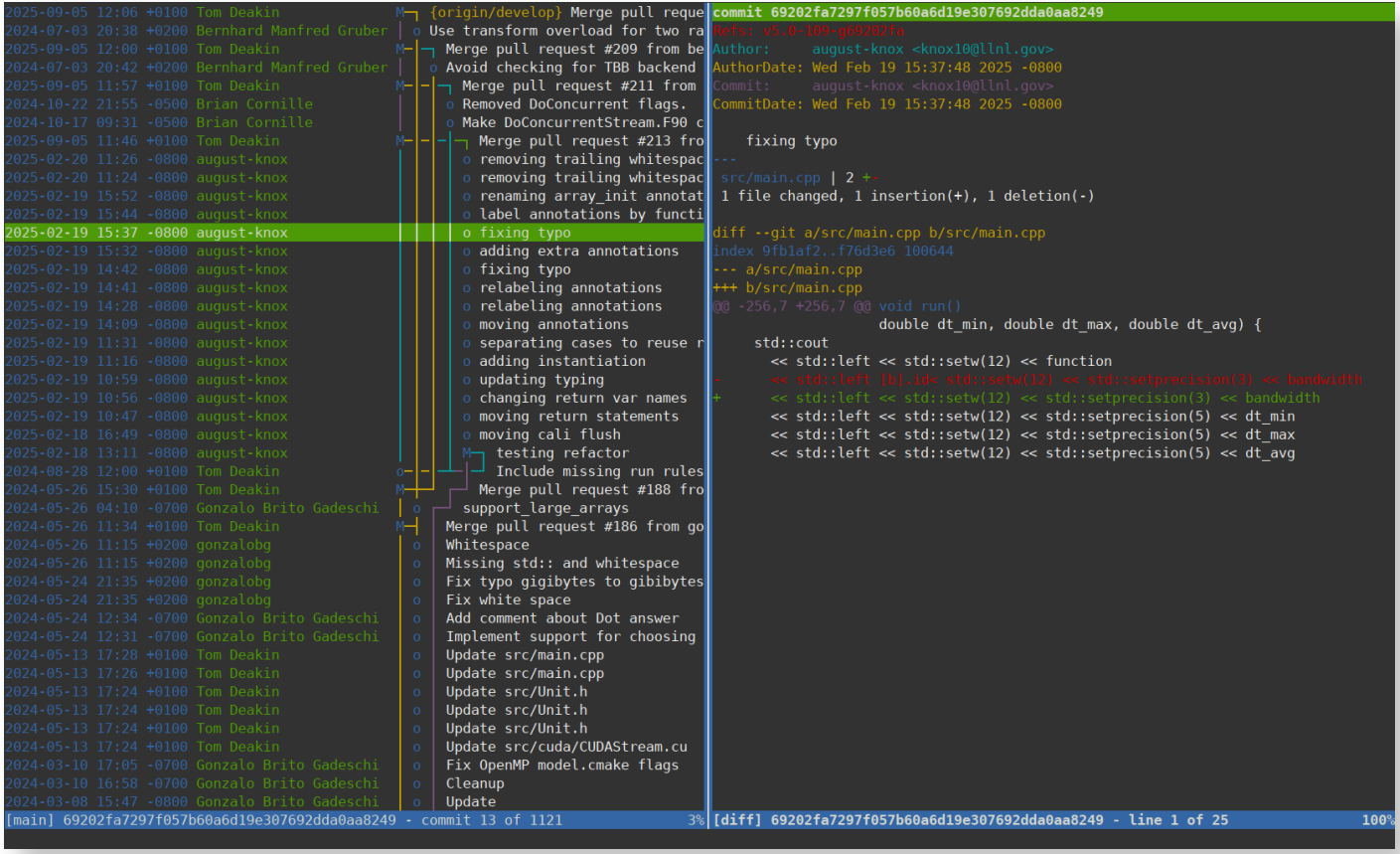
- `git fetch`
- `git rebase` **`origin/mybranch`**

I highly suggest to use git pull with rebase enabled

Useful commands - tig

Interactively browse history

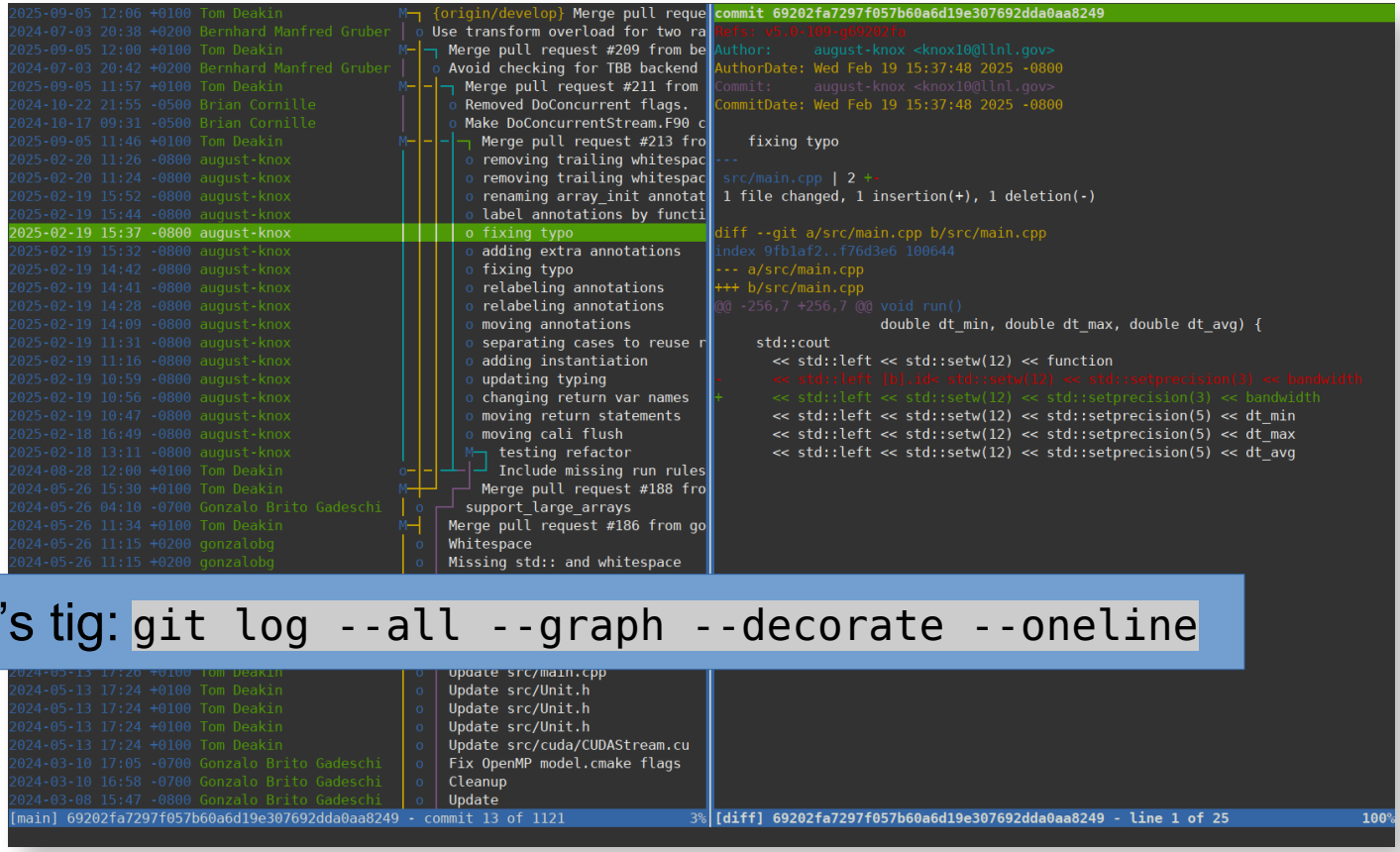
Use --all to see all branches



Useful commands - tig

Interactively browse history

Use --all to see all branches



Poor man's tig: `git log --all --graph --decorate --oneline`

Git history (interactive rebase)

Rebase can do much more than reordering commits:

- Edit/fix commits
- Delete commits
- Insert new commits
- Merge/split commits

Git history (interactive rebase)

Rebase can do much more than reordering commits:

- Edit/fix commits
- Delete commits
- Insert new commits
- Merge/split commits

→ `git rebase -i 0d8fdf1`: Edit history down to (excluding) commit **0d7fdf1**

Git history (interactive rebase)

Rebase can do much more than reordering commits:

- Edit/fix commits
- Delete commits
- Insert new commits
- Merge/split commits

→ `git rebase -i 0d8fdf1`: Edit history down to (excluding) commit **0d7fdf1**

→ `git rebase -i HEAD~8`: Edit history of previous 8 commits

Git history (interactive rebase)

Rebase can do much more than reordering commits:

- Edit/fix commits
- Delete commit
- Insert new com
- Merge/split co

→ `git rebase -i`

→ `git rebase -i`

```
1 pick c07e433b # rocmon: Remove obsolete destroyMarkerFileRocm
2 pick f7fa60ab # rocmon: Remove debug printf
3 edit 82fa47ce # rocmon: Fix memory leak in Lua API
4 pick b1a81528 # rocmon: Fix error handling for memory allocation
5 pick 37d6f907 # rocmon: Do not fail if metric cannot be calculated
6 pick 5f8bbcd8 # rocmon: Fix new counter last/full counting
7 pick 2e196ee2 # rocmon: Fix agent selection
8 pick fbf01bb3 # rocmon: Print more detailed device info
9
10 # Rebase fce4598b..fbf01bb3 onto fce4598b (8 commands)
11 #
12 # Commands:
13 # p, pick <commit> = use commit
14 # r, reword <commit> = use commit, but edit the commit message
15 # e, edit <commit> = use commit, but stop for amending
16 # s, squash <commit> = use commit, but meld into previous commit
17 # f, fixup [-C | -c] <commit> = like "squash" but keep only the previ
```

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`
- This also works for merging, e.g. during a conflict, via `git merge --abort`

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`
- This also works for merging, e.g. during a conflict, via `git merge --abort`

If you have already completed the rebase, use the reflogs for recovery.

- Reflogs contain previous commits (even “deleted” commits).

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`
- This also works for merging, e.g. during a conflict, via `git merge --abort`

If you have already completed the rebase, use the reflogs for recovery.

- Reflogs contain previous commits (even “deleted” commits).
- `git reflog`

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`
- This also works for merging, e.g. during a conflict, via `git merge --abort`

If you have already completed the rebase, use the reflogs for recovery.

- Reflogs contain previous commits (even “deleted” commits).
- `git reflog`
- `git reset --hard 84d72f1`

Git history (interactive rebase)

“I messed something up during my rebase. What do I do?”

- You can always abort a rebase with `git rebase --abort`
- This also works for merging, e.g. during a conflict, via `git merge --abort`

If you have already completed the rebase, use the reflogs for recovery.

- Reflogs contain previous commits (even “deleted” commits).
- `git reflog`
- `git reset --hard 84d72f1`
- Caution: “deleted” commits expire eventually

Git history (“clean” commits)

Tips:

- “I forgot something”: You can update (without creating a new one) the last commit via: `git commit --amend`.

Git history (“clean” commits)

Tips:

- “I forgot something”: You can update (without creating a new one) the last commit via: `git commit --amend`.
- Abort committing by leaving the commit message blank or clearing it.

Git history (“clean” commits)

Tips:

- “I forgot something”: You can update (without creating a new one) the last commit via: `git commit --amend`.
- Abort committing by leaving the commit message blank or clearing it.
- Use verbose `git commit`. Either via -v or `git config --global commit.verbose true`.

Git history (“clean” commits)

Tips:

- “I forgot something”: You can update (without creating a new one) the last commit via: `git commit --amend`.
- Abort committing by leaving the commit message blank or clearing it.
- Use verbose `git commit`. Either via -v or `git config --global commit.verbose true`.
- My opinion: Avoid git commit -m “…”. Review your commits via verbose commit.

Git history (“clean” commits)

Tips:

- “I forgot somethi
  last commit via: `g`
- Abort committing
- Use verbose `git`
  `commit.verbose t`
- My opinion: Avoi
  commit.

```
1  Added more text.
2
3  The paragraph after the first line gives you opportunity to explain your
4  commit in more detail. Why did you change things?
5
6  # Please enter the commit message for your changes. Lines starting
7  # with '#' will be ignored, and an empty message aborts the commit.
8  #
9  # Date:      Mon Feb 2 16:51:53 2026 +0100
10 #
11 # On branch master
12 # Changes to be committed:
13 #       modified:   myfile
14 #
15 # ------------------------ >8 ------------------------
16 # Do not modify or remove the line above.
17 # Everything below it will be ignored.
18 diff --git a/myfile b/myfile
19 index 1756cf9..5427306 100644
20 --- a/myfile
21 +++ b/myfile
22 @@ -1 +1,3 @@
23  Hello people.
24 +
25 +Nobody there.
```

Git history (conclusion)

Don’t be afraid of rewriting/rebasing history!

Git history (conclusion)

Don’t be afraid of rewriting/rebasing history!

Unless it is already public!

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Git staging area

Git staging area

“I want to commit. I have to type git add file.txt and git commit”:

Git staging area

“I want to commit. I have to type git add file.txt and git commit”:

- Yes, but why? Not all version control systems have it (e.g. SVN)

Git staging area

“I want to commit. I have to type git add file.txt and git commit”:

- Yes, but why? Not all version control systems have it (e.g. SVN)
- Before we commit we can carefully choose what to commit!

Git staging area

"I want to commit. I have to type git add file.txt and git commit":

- Yes, but why? Not all version control systems have it (e.g. SVN)
- Before we commit we can carefully choose what to commit!

```
michael@michael-HP♯/tmp/mygit3♪ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   myfile2

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   myfile

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        myfile3
```

Git staging area

"I want to commit. I have to type git add file.txt and git commit":

- Yes, but why? Not all version control systems have it (e.g. SVN)
- Before we commit we can carefully choose what to commit!

```
michael@michael-HP♯/tmp/mygit3♪ git status
On michael@michael-HP♯/tmp/mygit3♪ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   myfile
        modified:   myfile2

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   myfile2

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        myfile3
```

Git staging area

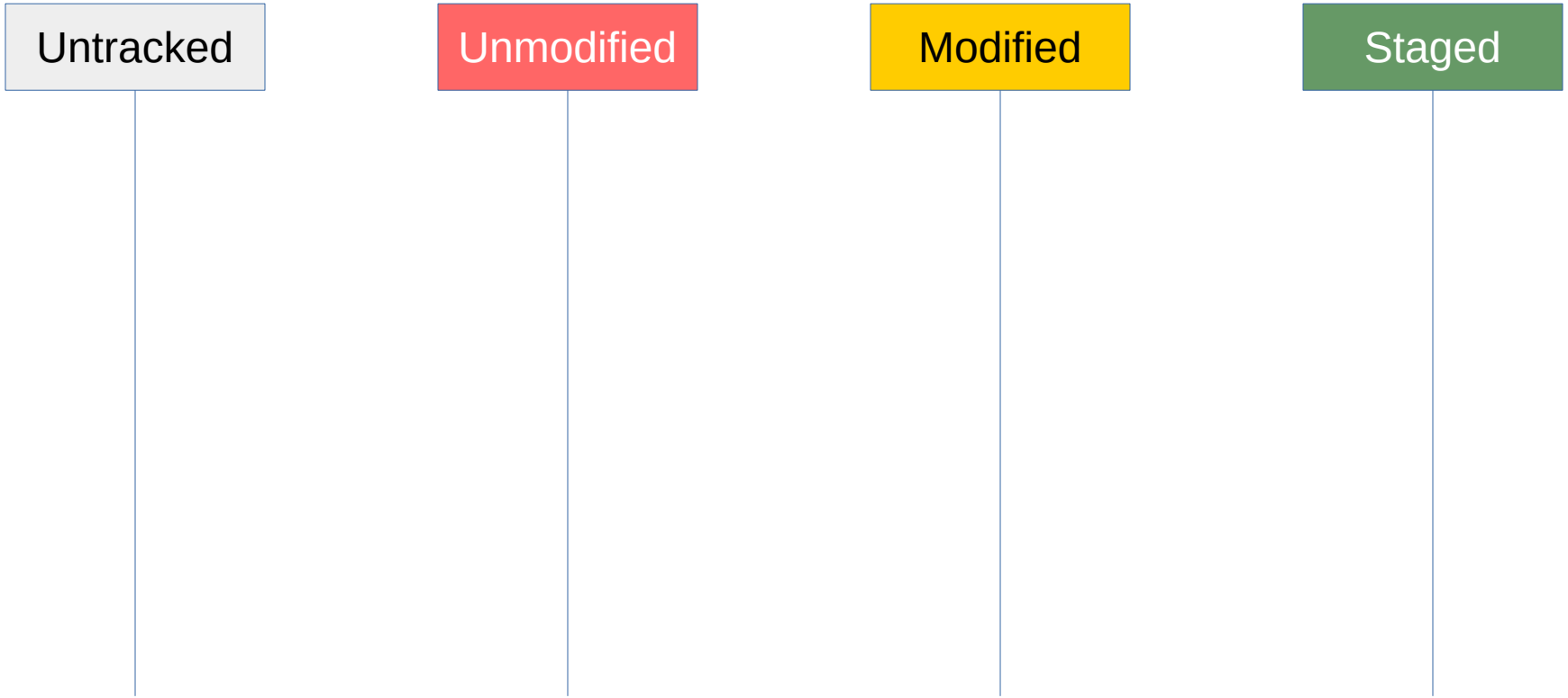


Image inspired by Pro Git chapter 2.2

Git staging area

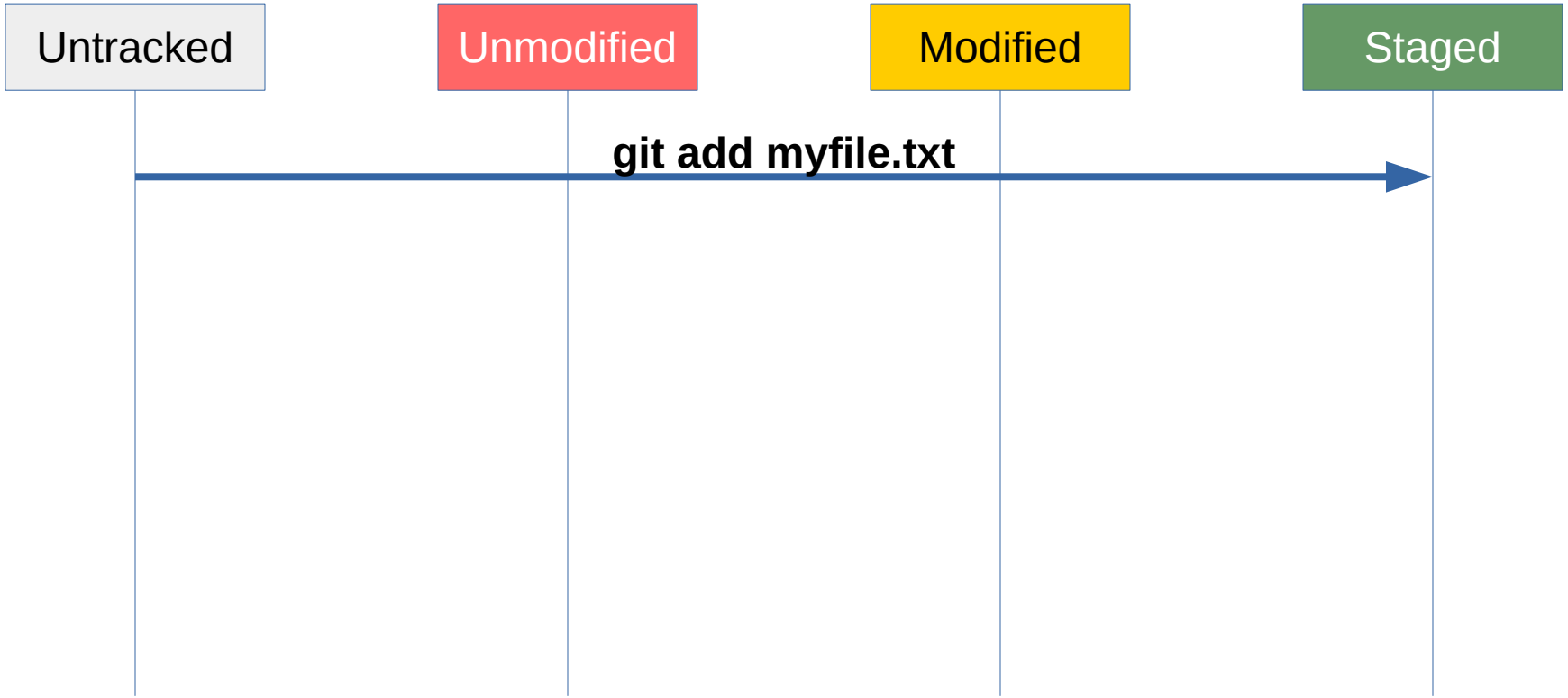


Image inspired by Pro Git chapter 2.2

Git staging area

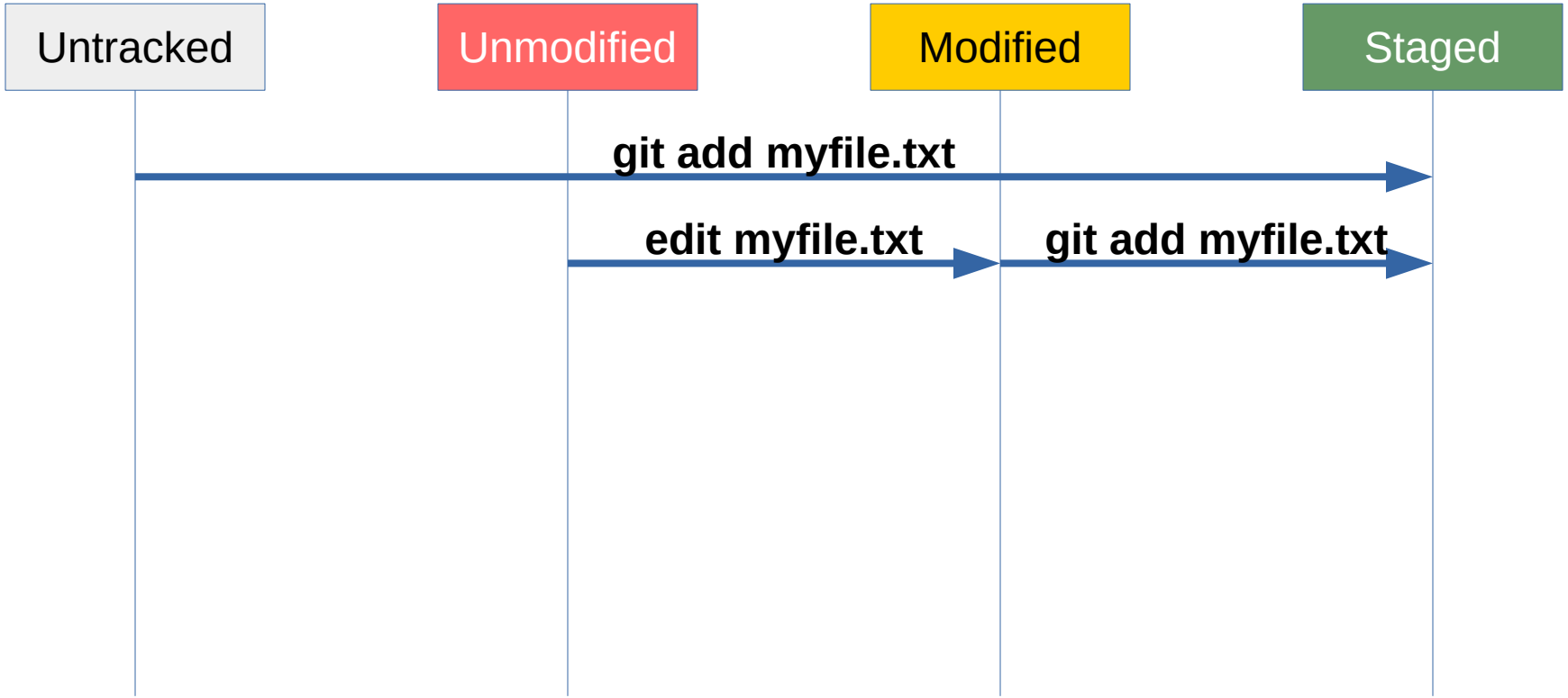


Image inspired by Pro Git chapter 2.2

Git staging area

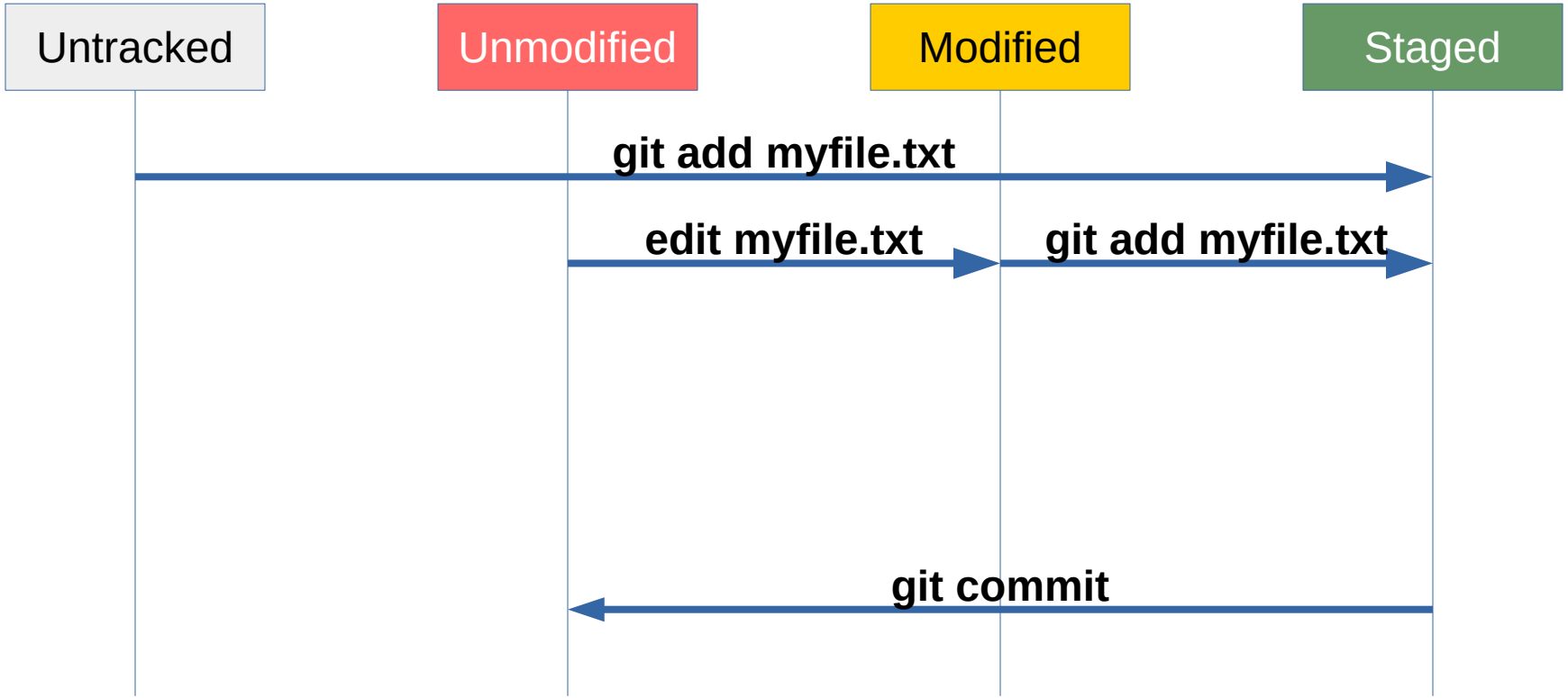


Image inspired by Pro Git chapter 2.2

Git staging area

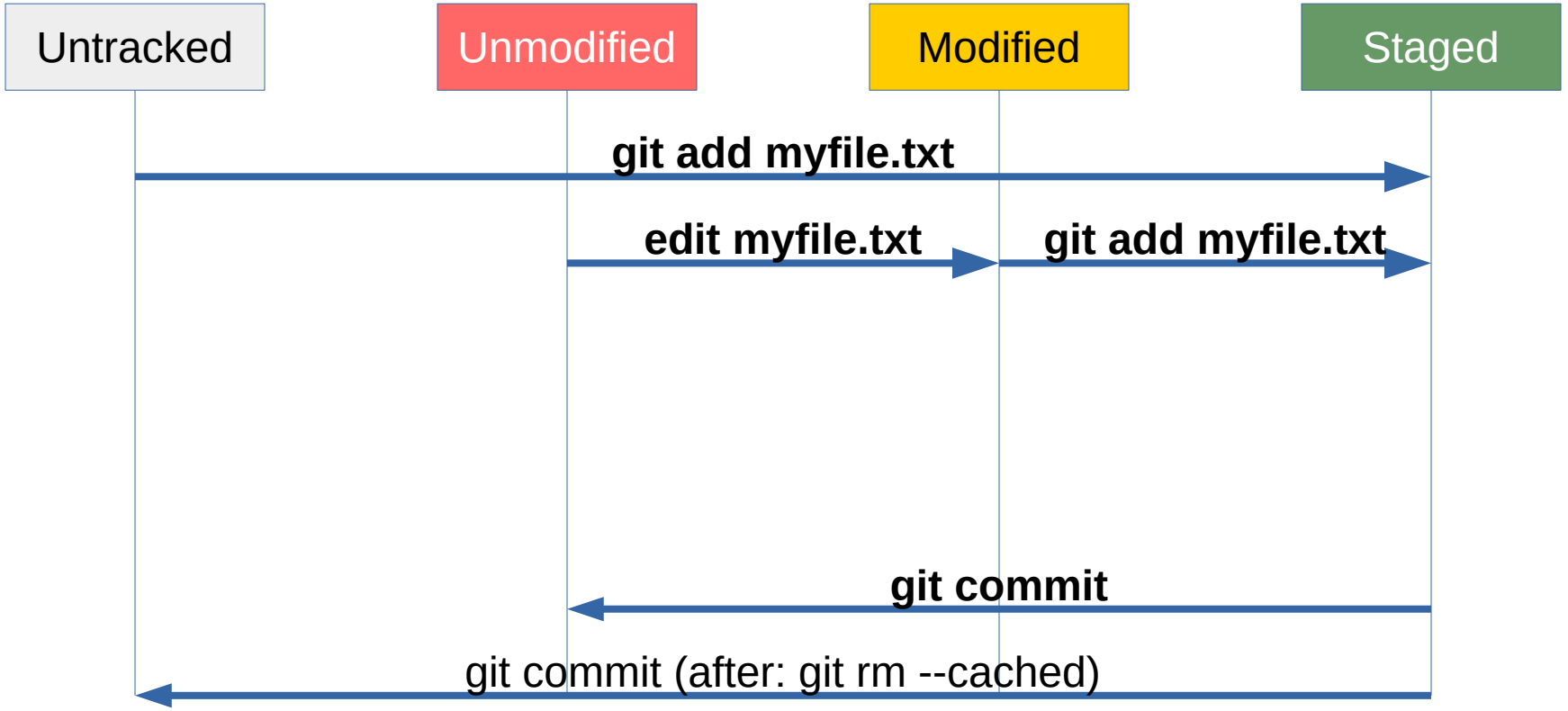


Image inspired by Pro Git chapter 2.2

Git staging area

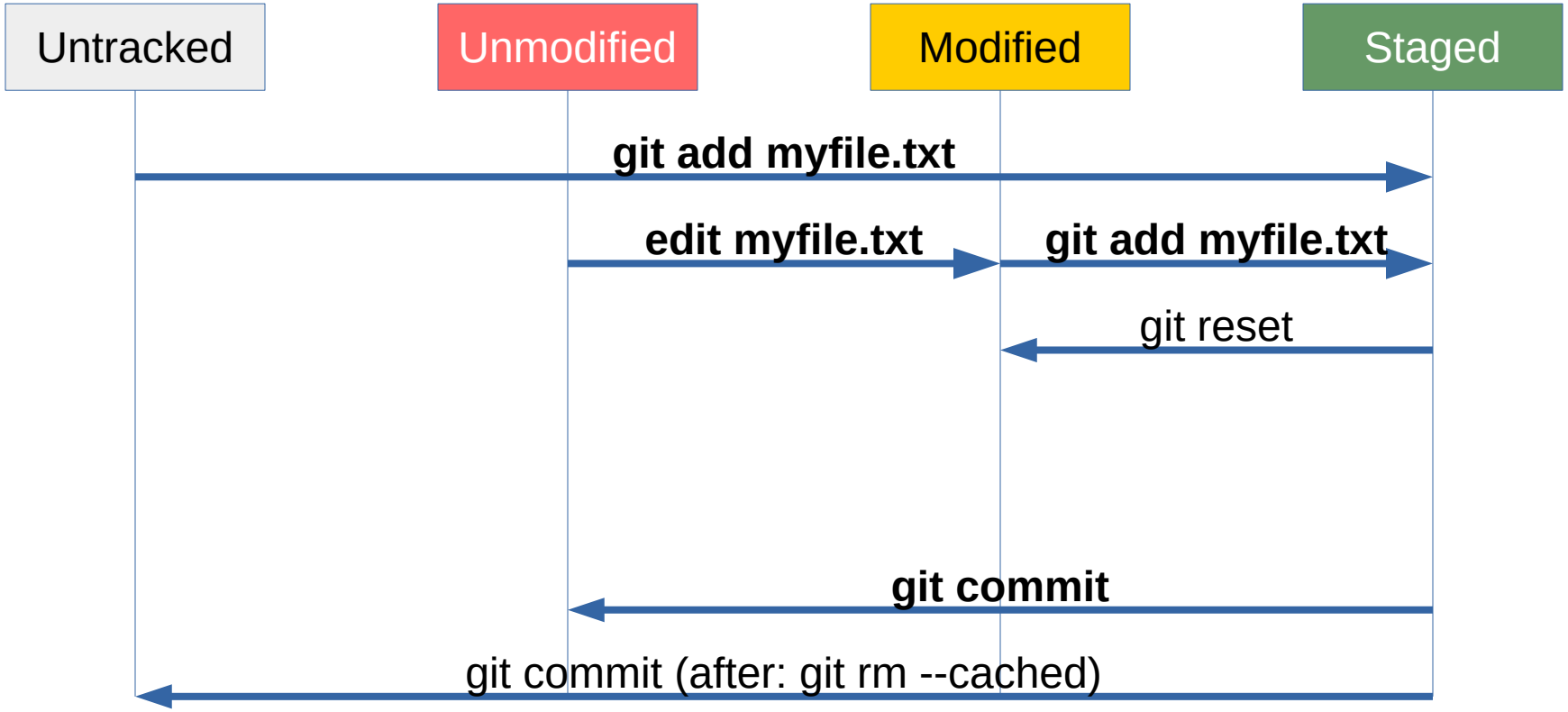


Image inspired by Pro Git chapter 2.2

Git staging area

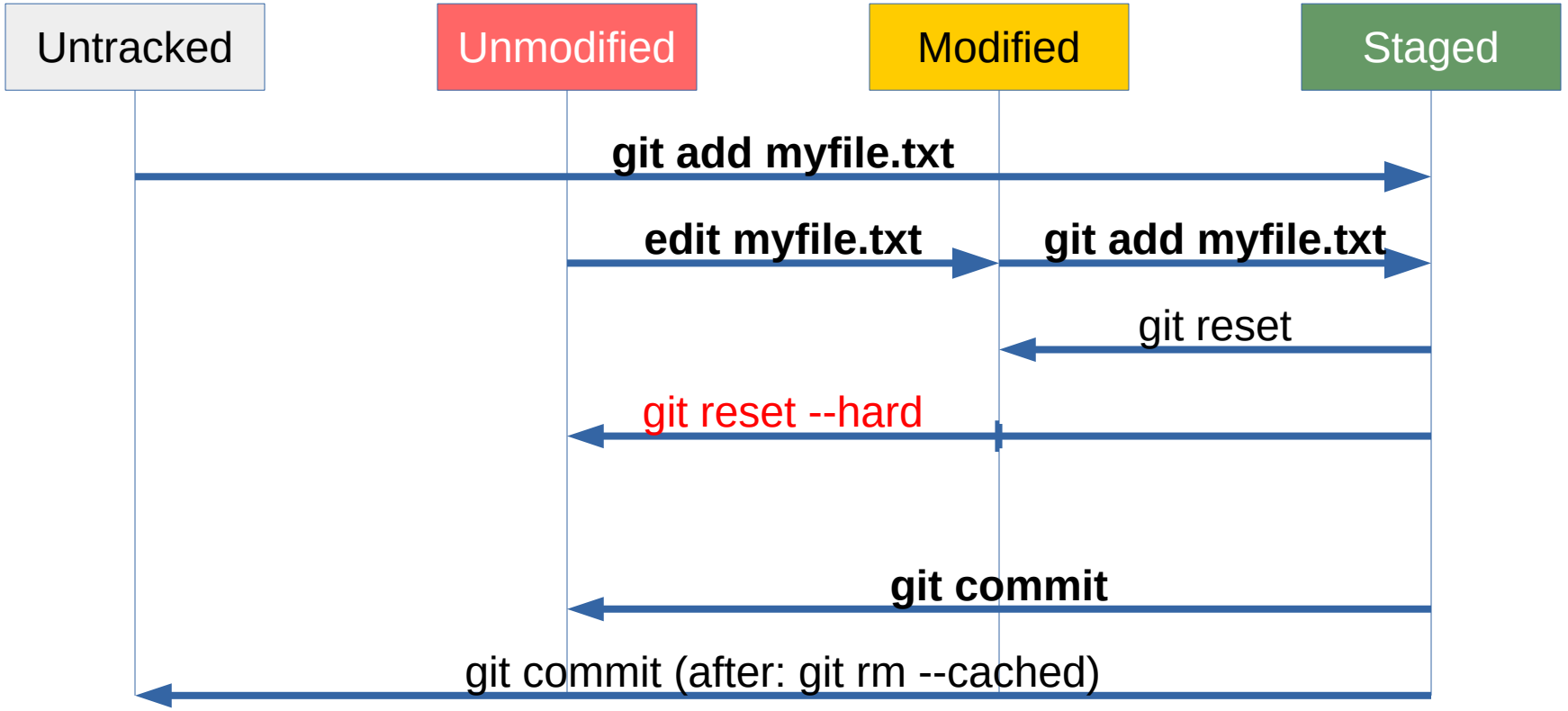


Image inspired by Pro Git chapter 2.2

Git staging area

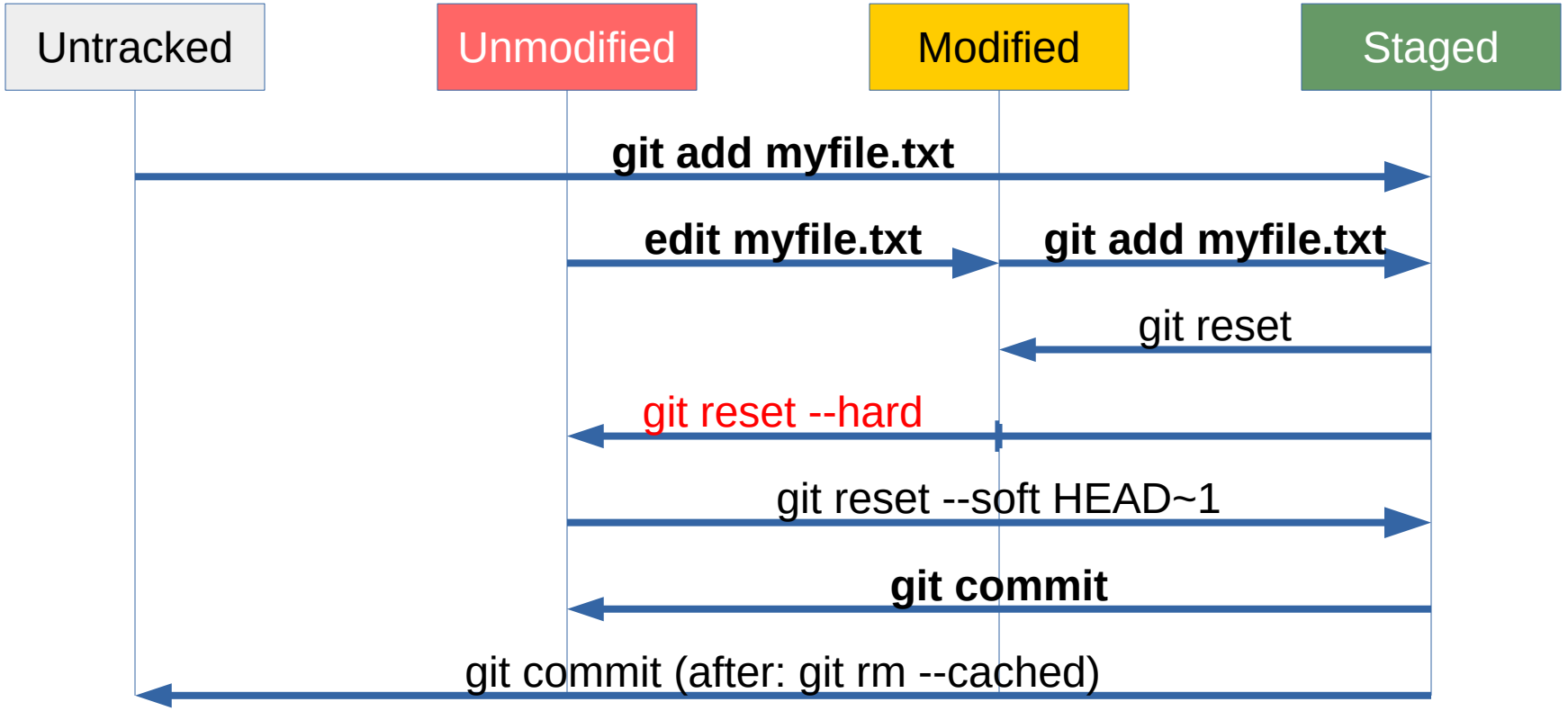


Image inspired by Pro Git chapter 2.2

Git staging area

“But I only want to commit funcA in main.c, not my debug statements”:

Git staging area

“But I only want to commit funcA in main.c, not my debug statements”:

→ `git add -p main.c`

Git staging area

“But I only want to commit funcA”

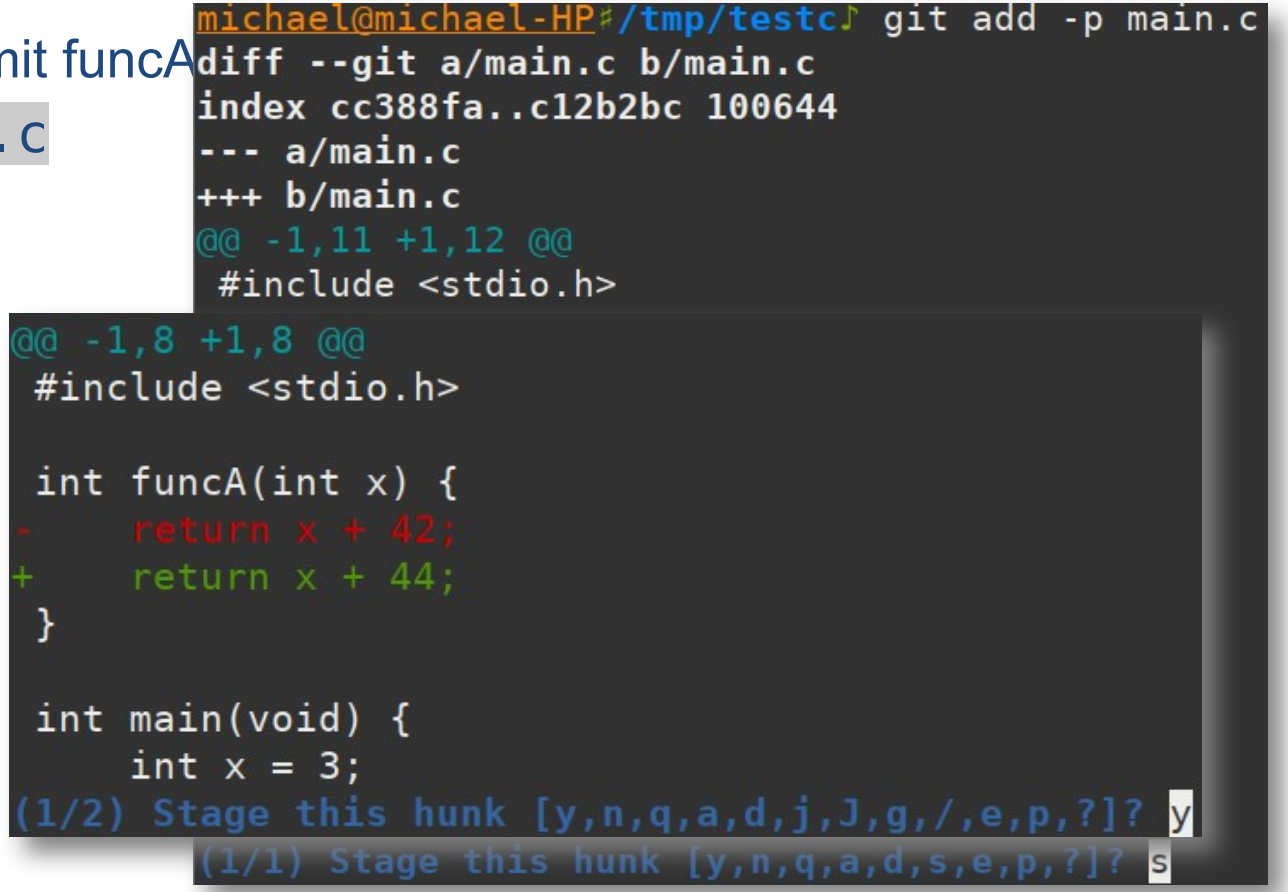
→ `git add -p main.c`

```
michael@michael-HP♯/tmp/testc♪ git add -p main.c
diff --git a/main.c b/main.c
index cc388fa..c12b2bc 100644
--- a/main.c
+++ b/main.c
@@ -1,11 +1,12 @@
 #include <stdio.h>

 int funcA(int x) {
-    return x + 42;
+    return x + 44;
 }

 int main(void) {
     int x = 3;
+    printf("debug: \d\n", x);
     x = funcA(x);
     printf("a: \d\n", x);
 }
(1/1) Stage this hunk [y,n,q,a,d,s,e,p,?]? s
```

Git staging area

“But I only want to commit funcA”

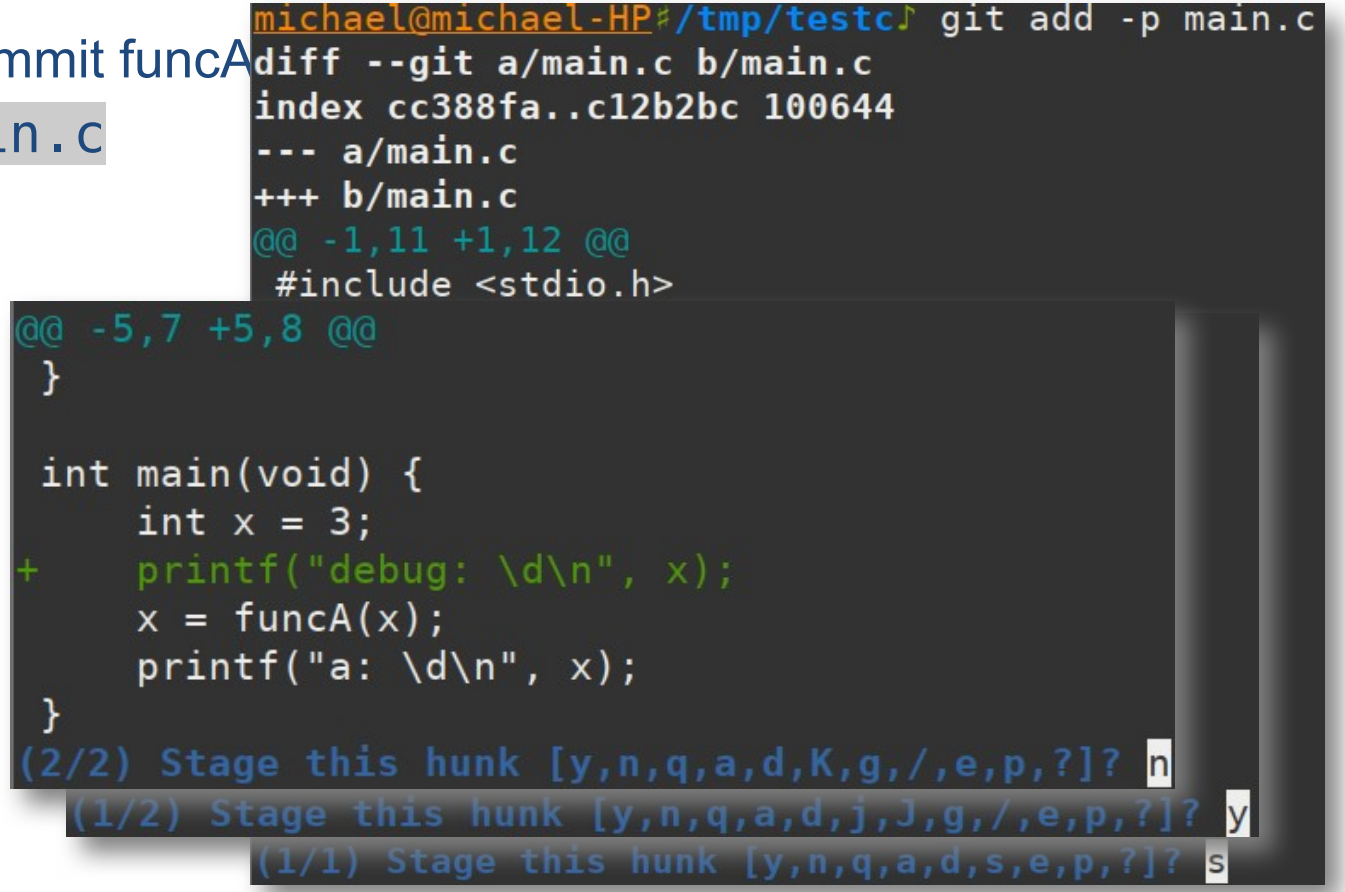
→ `git add -p main.c`



Git staging area

“But I only want to commit funcA

→ `git add -p main.c`



NHR FAU

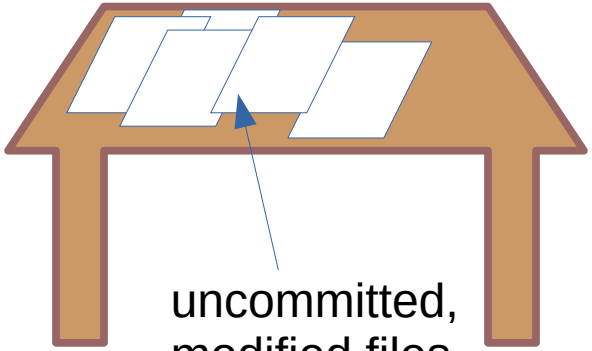
FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Git stash

git stash

```
unrz104h@testfront1♯~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

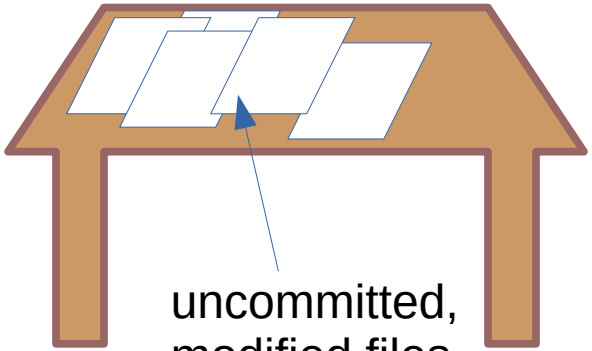
Current branch: **main**



git stash

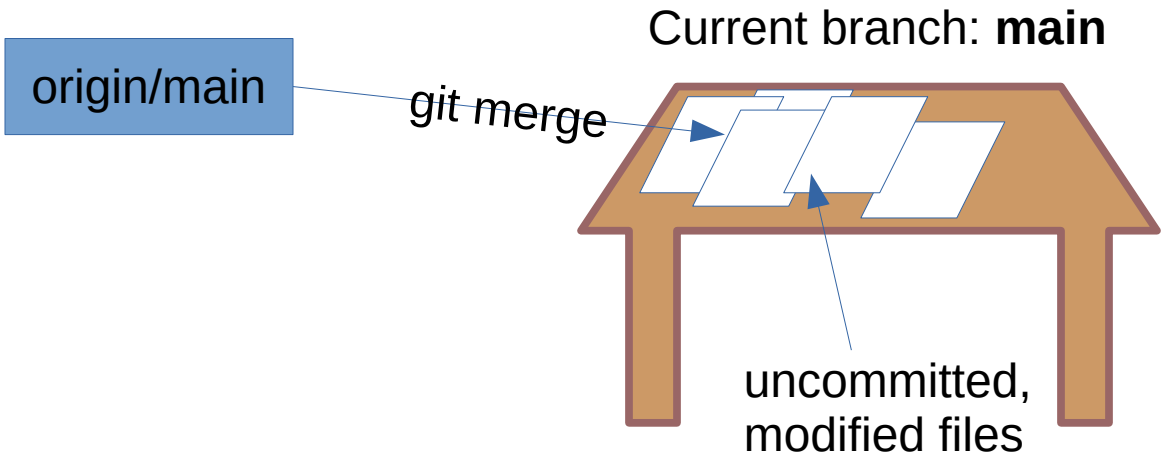
```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Current branch: **main**



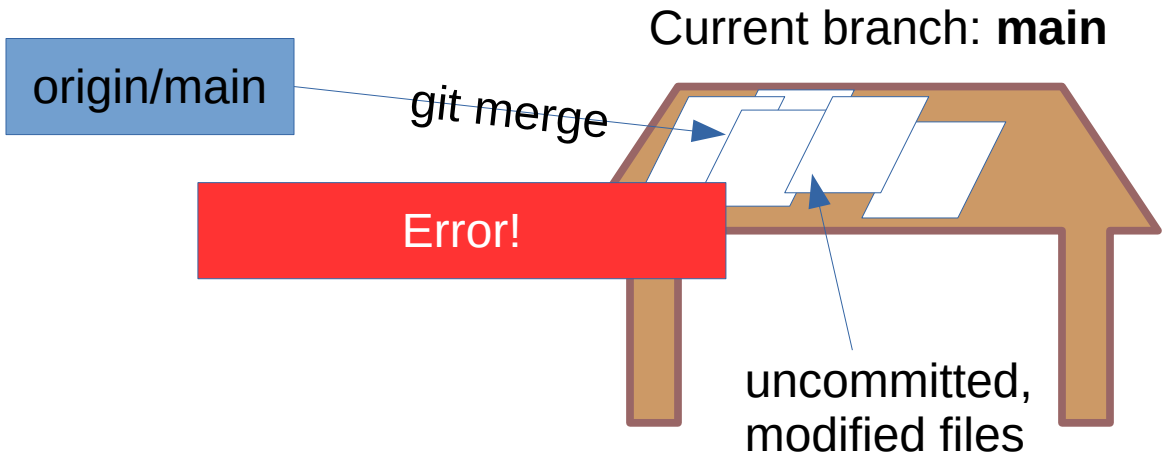
git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```



git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

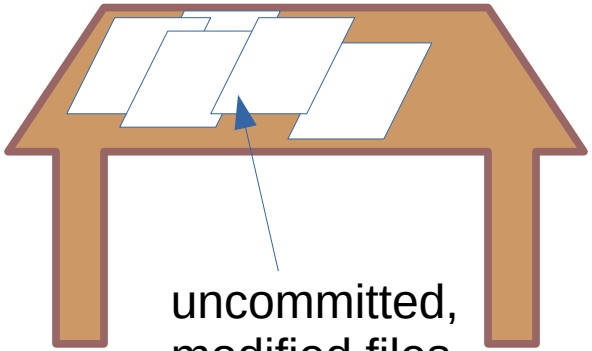


git stash

```
unrz104h@testfront1♯~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

origin/main

Current branch: **main**



uncommitted,
modified files

git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Solution!

origin/main

Current branch: **main**

stash@{0}

...

git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Solution!

origin/main

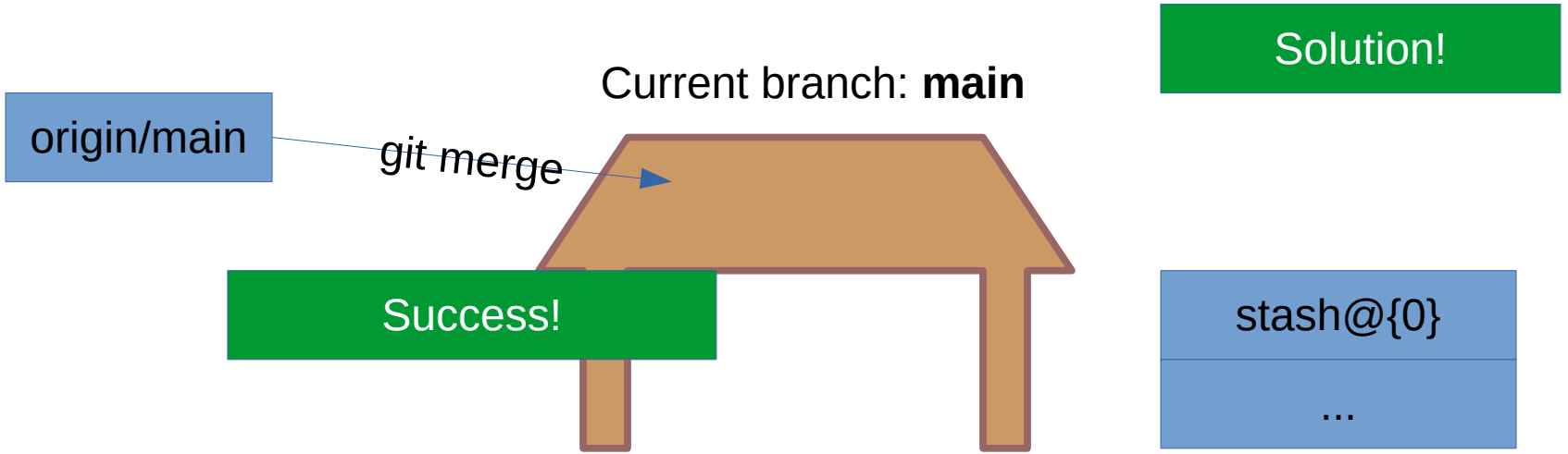
Current branch: **main**

stash@{0}

...

git stash

```
unrz104h@testfront1:~/Projects/rocm-systems$ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```



git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Solution!

Current branch: **main**

stash@{0}

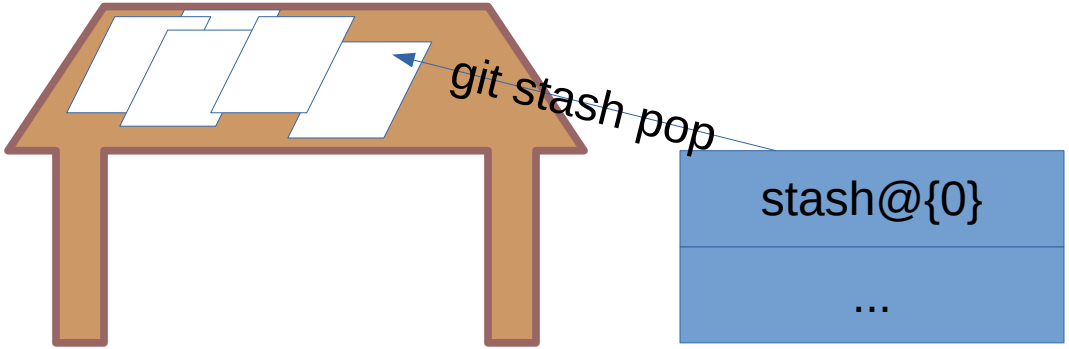
...

git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Solution!

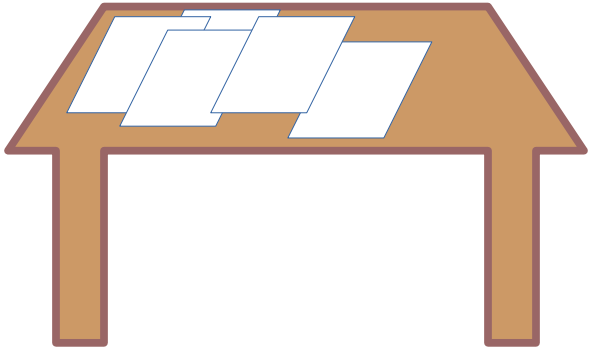
Current branch: **main**



git stash

```
unrz104h@testfront1#~/Projects/rocm-systems♪ git pull
Updating e45c56c0f8..66ee941fea
error: Your local changes to the following files would be overwritten by merge:
        README.md
Please commit your changes or stash them before you merge.
Aborting
```

Current branch: **main**



Solution!

...

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Git remotes

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone ‘https://github.com/RRZE-HPC/likwid’`:

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone ‘https://github.com/RRZE-HPC/likwid’`:

- `mkdir likwid && cd likwid`

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone ‘https://github.com/RRZE-HPC/likwid’`:

- `mkdir likwid && cd likwid`
- `git init`

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone ‘https://github.com/RRZE-HPC/likwid’`:

- `mkdir likwid && cd likwid`
- `git init`
- `git remote add origin ‘https://github.com/RRZE-HPC/likwid’`

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone ‘https://github.com/RRZE-HPC/likwid’`:

- `mkdir likwid && cd likwid`
- `git init`
- `git remote add origin ‘https://github.com/RRZE-HPC/likwid’`
- `git switch --track origin/master`

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone 'https://github.com/RRZE-HPC/likwid'`:

- `mkdir likwid && cd likwid`
- `git init`
- `git remote add origin 'https://github.com/RRZE-HPC/likwid'`
- `git switch --track origin/master`

→ **origin** is Git’s default name for a remote

Git remotes

“I want to push to my GitHub/GitLab. This is what **origin** is, right?”:

Equivalent of `git clone 'https://github.com/RRZE-HPC/likwid'`:

- `mkdir likwid && cd likwid`
- `git init`
- `git remote add origin 'https://github.com/RRZE-HPC/likwid'`
- `git switch --track origin/master`

→ **origin** is Git’s default name for a remote

→ **master** is the default branch of LIKWID

Git remotes

Interact with repositories outside yours:

- Way of interacting with GitHub / GitLab

Git remotes

Interact with repositories outside yours:

- Way of interacting with GitHub / GitLab
- “Fancy platforms” are not mandatory. Remote can also be:
  - A folder on a machine reachable via standard SSH

Git remotes

Interact with repositories outside yours:

- Way of interacting with GitHub / GitLab
- “Fancy platforms” are not mandatory. Remote can also be:
  - A folder on a machine reachable via standard SSH
  - A different folder on your local machine

Git remotes

Interact with repositories outside yours:

- Way of interacting with GitHub / GitLab
- “Fancy platforms” are not mandatory. Remote can also be:
  - A folder on a machine reachable via standard SSH
  - A different folder on your local machine
- You can have as many remotes as you like (e.g. mirrors, backups, etc.)

Git remotes

Example: Copy **main** branch from Github repository to GitLab:

- `git clone ‘git@github.com:myuser/myproject.git’`

Git remotes

Example: Copy **main** branch from Github repository to GitLab:

- `git clone ‘git@github.com:myuser/myproject.git’`
- `cd myproject`

Git remotes

Example: Copy **main** branch from Github repository to GitLab:

- `git clone ‘git@github.com:myuser/myproject.git’`
- `cd myproject`
- `git remote add **gitlab** ‘git@gitlab.com:myuser/myproject.git’`

Git remotes

Example: Copy **main** branch from Github repository to GitLab:

- `git clone ‘git@github.com:myuser/myproject.git’`
- `cd myproject`
- `git remote add gitlab ‘git@gitlab.com:myuser/myproject.git’`
- `git push gitlab main`

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

- Branch “tracking” assigns a local branch to a remote

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

- Branch “tracking” assigns a local branch to a remote
- `git clone` automatically tracks the remote’s default branch (**origin/HEAD**)

  E.g. **main** automatically tracks **origin/main**

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

- Branch “tracking” assigns a local branch to a remote
- `git clone` automatically tracks the remote’s default branch (**origin/HEAD**)

  E.g. **main** automatically tracks **origin/main**
- You can change it any time: `git branch -u neworigin`

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

- Branch “tracking” assigns a local branch to a remote
- `git clone` automatically tracks the remote’s default branch (**origin/HEAD**)

  E.g. **main** automatically tracks **origin/main**
- You can change it any time: `git branch -u neworigin`
- When it doesn’t exist yet on remote: `git push -u neworigin`

Git remotes

“Why do I sometimes need to specify the remote and sometimes not?”:

- Branch “tracking” assigns a local branch to a remote
- `git clone` automatically tracks the remote’s default branch (**origin/HEAD**)

  E.g. **main** automatically tracks **origin/main**
- You can change it any time: `git branch -u neworigin`
- When it doesn’t exist yet on remote: `git push -u neworigin`
- Afterwards you can push/pull “normally”

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Git summary

Summary

Useful commands to remember:

- `git status` → Know what's going on
- `git log -p <filename>` → Show history of a file
- `git log -S <searchterm>` → Search for diff affecting a term
- `git log -G <searchterm>` → Search for diff containing a term
- `git blame <filename>` → Show file with history annotation
- `git reflog` → Show all commits (including "lost" commits)
- `git gc` → Cleanup unreferenced commits (reflog recovery impossible!)
- `git rebase <branch>` → Reapply commits onto *branch*
- `git rebase -i <commit>` → Interactively edit commits up to *commit*
- `git cherry-pick <commit>` → Reapply *commit* to current branch

Summary

Useful commands to remember:

- `git add -p <filename>` → Perform `git add` on parts of *filename*
- `git commit -v` → See what you commit
- `git stash` → Move unstaged changes to “stash” and reset files
- `git stash pop` → Restore all previous unstaged changes
- `git reset <filename>` → Unstage all changes of *filename*
- `git reset --hard` → Discard all uncommited changes
- `git reset --hard <commit>` → Make current branch point to *commit*. Discards all commits that are not part of new branch history!
- `git reset <commit>` → Show changes of commit
- `git bisect start <badcommit> <goodcommit>` → Find a regression between two commits

Summary

Other useful resources:

- Manpages are really good!
- `man gitrevisions` → Explanations how version strings look like
- Pro Git book