Git in real life

Alessandro Sciarra

Z02 – Software Development Center

XX December 2022

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

CRC-TR 211
Strong-interaction matter under extreme conditions

Outline of the talk

1. Rewriting history
2. Git rebase
3. Git tag
4. Semantic versioning
5. Gitflow

Rewriting history

About rewriting history

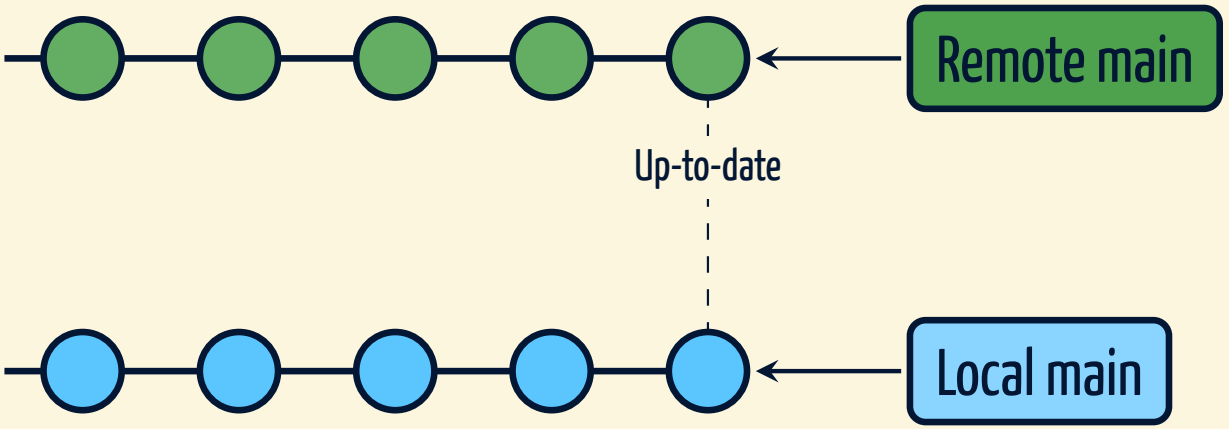
Yes, you can change the history of a repository!

- Very bad if there are different copies of the history { E.g. branches, remote(s) }
- If you rewrite shared history,
  - it is generally hard to make the same change elsewhere and
  - merging (and hence pull/push) can lead to duplicated commits in history
- Fine as long as
  - yours is the only repository containing the rewritten history or
  - you work on a git project alone and you know what to do then

About rewriting history

Yes, you can change the history of a repository!

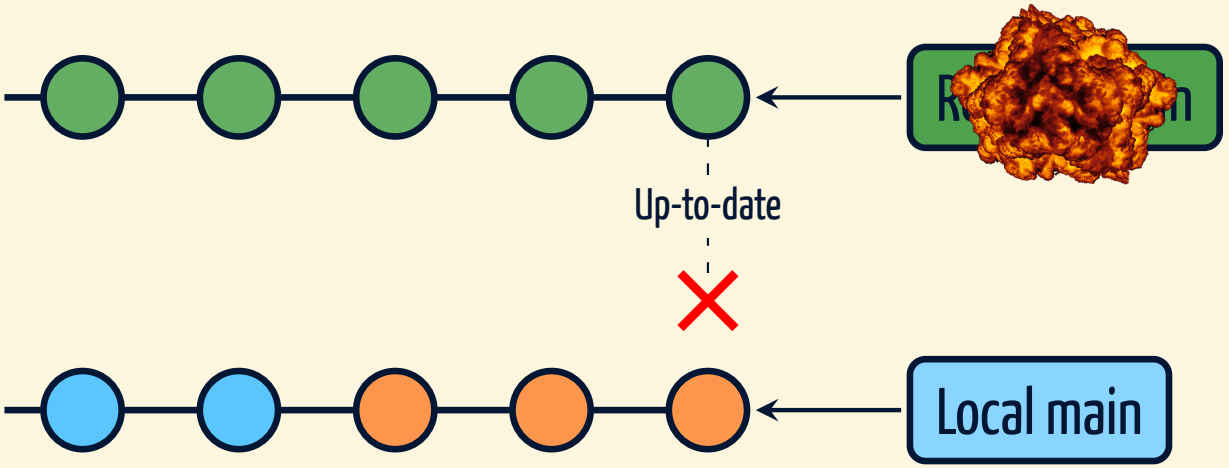
- Very bad if there are different copies of the history { E.g. branches, remote(s) }
- If you rewrite shared history,
  - it is generally hard to make the same change elsewhere and
  - merging (and hence pull/push) can lead to duplicated commits in history



About rewriting history

Yes, you can change the history of a repository!

- Very bad if there are different copies of the history { E.g. branches, remote(s) }
- If you rewrite shared history,
  - it is generally hard to make the same change elsewhere and
  - merging (and hence pull/push) can lead to duplicated commits in history



As long as you rewrite history contained elsewhere, you are in troubles!

Changing the last commit (message)

1. How can I fix a typo in the **last** commit message?

> **This procedure will change history!**
>
> Said differently, feel free to do it if you **did not** share history!

```
$ git commit -m "Added engine implementation"
[airplane f8b0c25] Added engine implementation
24 files changed, 1340 insertions(+), 476 deletions(-)
# Ops! I used a verb in past form, not conforming!
```


There is a nice official page about different situations in which rewriting history is needed.

Changing the last commit (message)

1. How can I fix a typo in the **last** commit message?

This procedure will change history!

Said differently, feel free to do it if you **did not** share history!

```
$ git commit -m "Added engine implementation"
[airplane f8b0c25] Added engine implementation
24 files changed, 1340 insertions(+), 476 deletions(-)
# Ops! I used a verb in past form, not conforming!
```

```
# With clean staging area:
$ git commit --amend -m "Add engine implementation"
[airplane 33f53f4] Add engine implementation
Date: Fri Nov 25 08:52:22 2018 +0100
24 files changed, 1340 insertions(+), 476 deletions(-)
```


There is a nice official page about different situations in which rewriting history is needed.

Changing the last commit (content)

2. I forgot some changes in the **last** commit! And now?

This procedure will change history!

Said differently, feel free to do it if you **did not** share history!

```
$ git commit -m "Review take-off system"
[airplane e1df32a] Review take-off system
1 file changed, 230 insertions(+), 61 deletions(-)
# Ops! I forgot a file!
```


There is a nice official page about different situations in which rewriting history is needed.

Changing the last commit (content)

2 I forgot some changes in the **last** commit! And now?

This procedure will change history!

Said differently, feel free to do it if you **did not** share history!

```
$ git commit -m "Review take-off system"
[airplane e1df32a] Review take-off system
1 file changed, 230 insertions(+), 61 deletions(-)
# Ops! I forgot a file!
```

```
$ git add wheels_electronic.h
$ git commit --amend --no-edit
[airplane c13e34f] Add engine implementation
Date: Fri Nov 25 08:45:18 2022 +0100
2 files changed, 345 insertions(+), 88 deletions(-)
```


There is a nice official page about different situations in which rewriting history is needed.

Changing the last commit (content)

2 I forgot some changes in the **last** commit! And now?

This procedure will change history!

Said differently, feel free to do it if you **did not** share history!

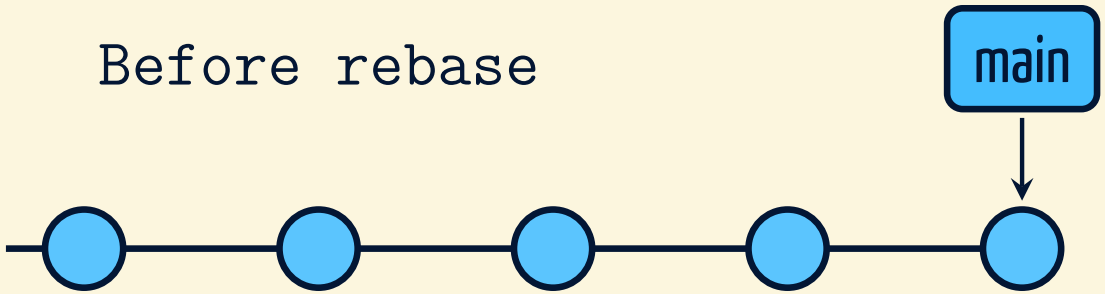
```
$ git commit -m "Review take-off system"
[airplane e1df32a] Review take-off system
1 file changed, 230 insertions(+), 61 deletions(-)
# Ops! I forgot a file!
```

```
$ git add wheels_electronic.h
$ git commit --amend -C HEAD # use original commit timestamp
[airplane c13e34f] Add engine implementation
Date: Fri Nov 25 08:33:01 2022 +0100
2 files changed, 345 insertions(+), 88 deletions(-)
```


There is a nice official page about different situations in which rewriting history is needed.

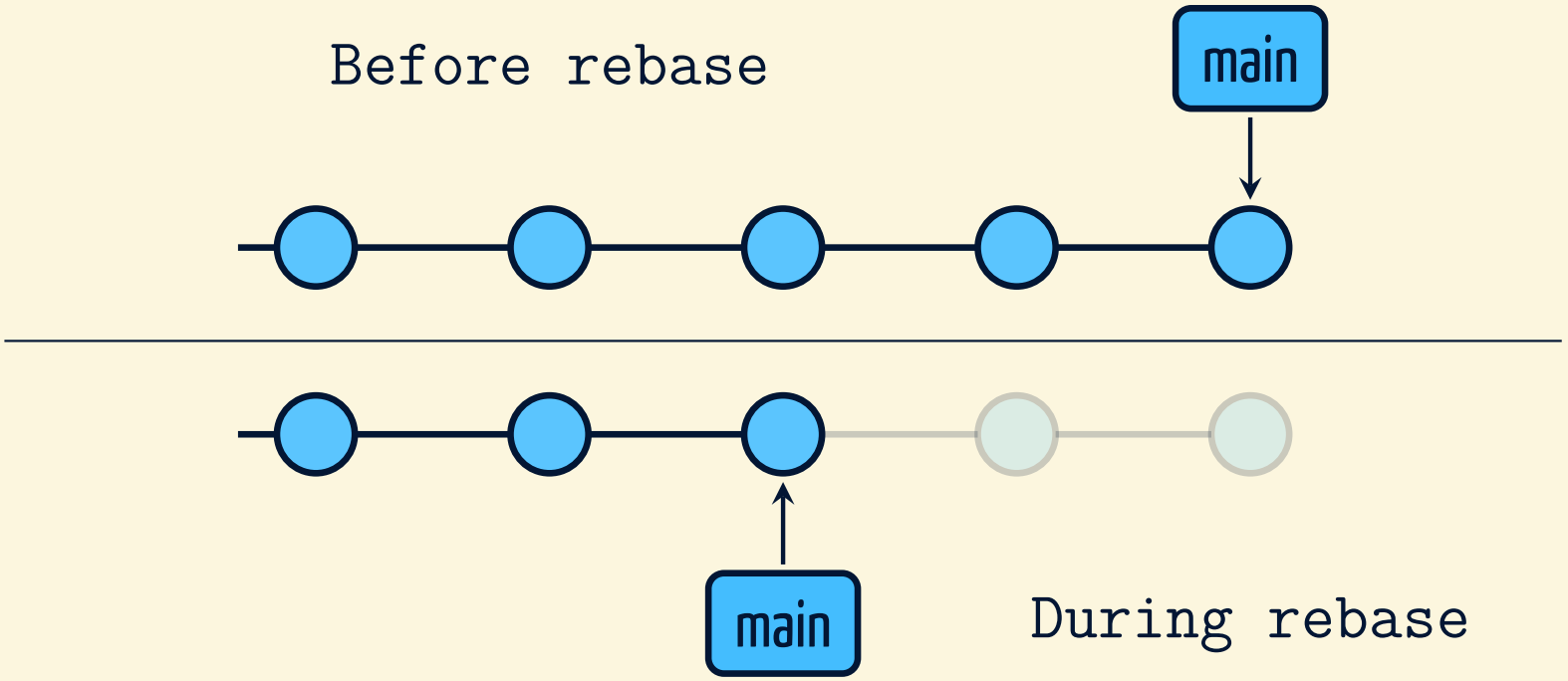
Git rebase

Which is the abstract idea of a rebase?



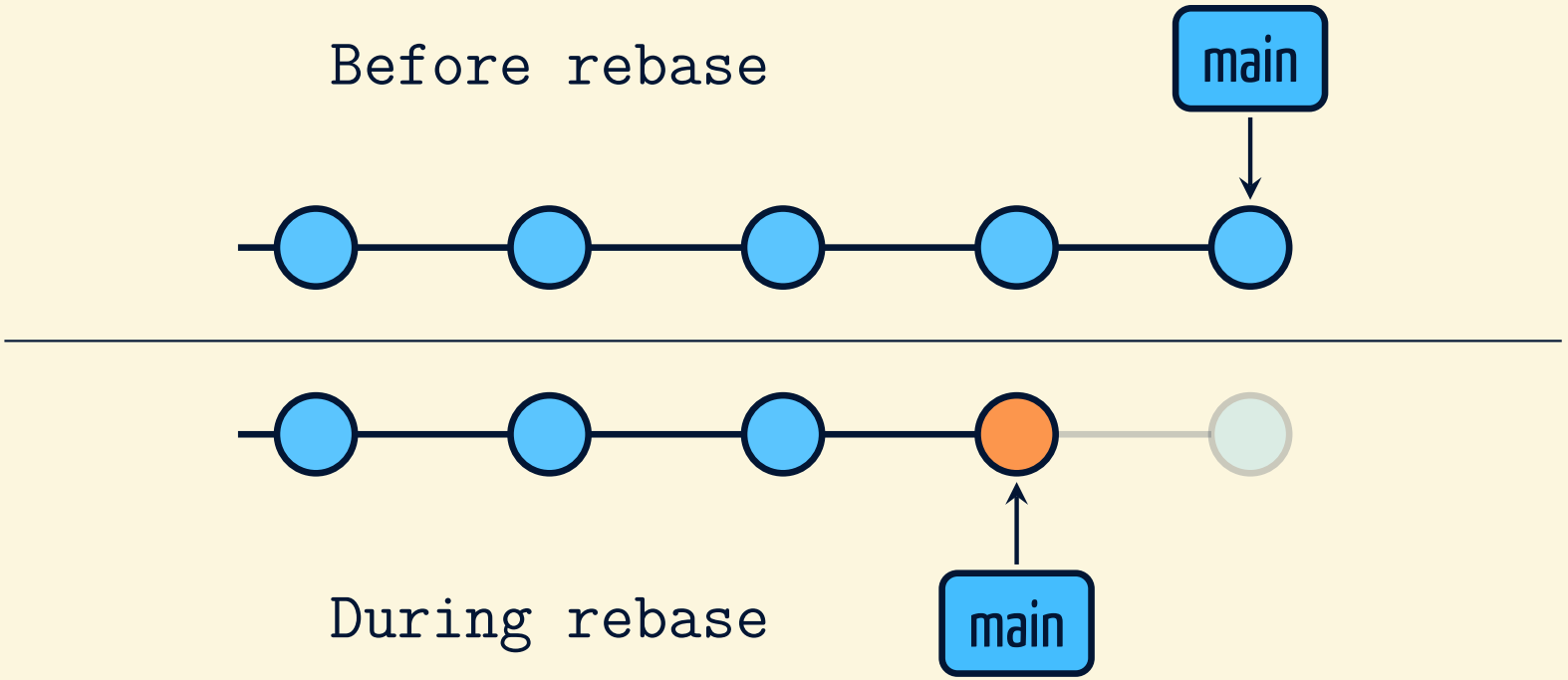
Which is the abstract idea of a rebase?

1. Go back in history till a **given** point



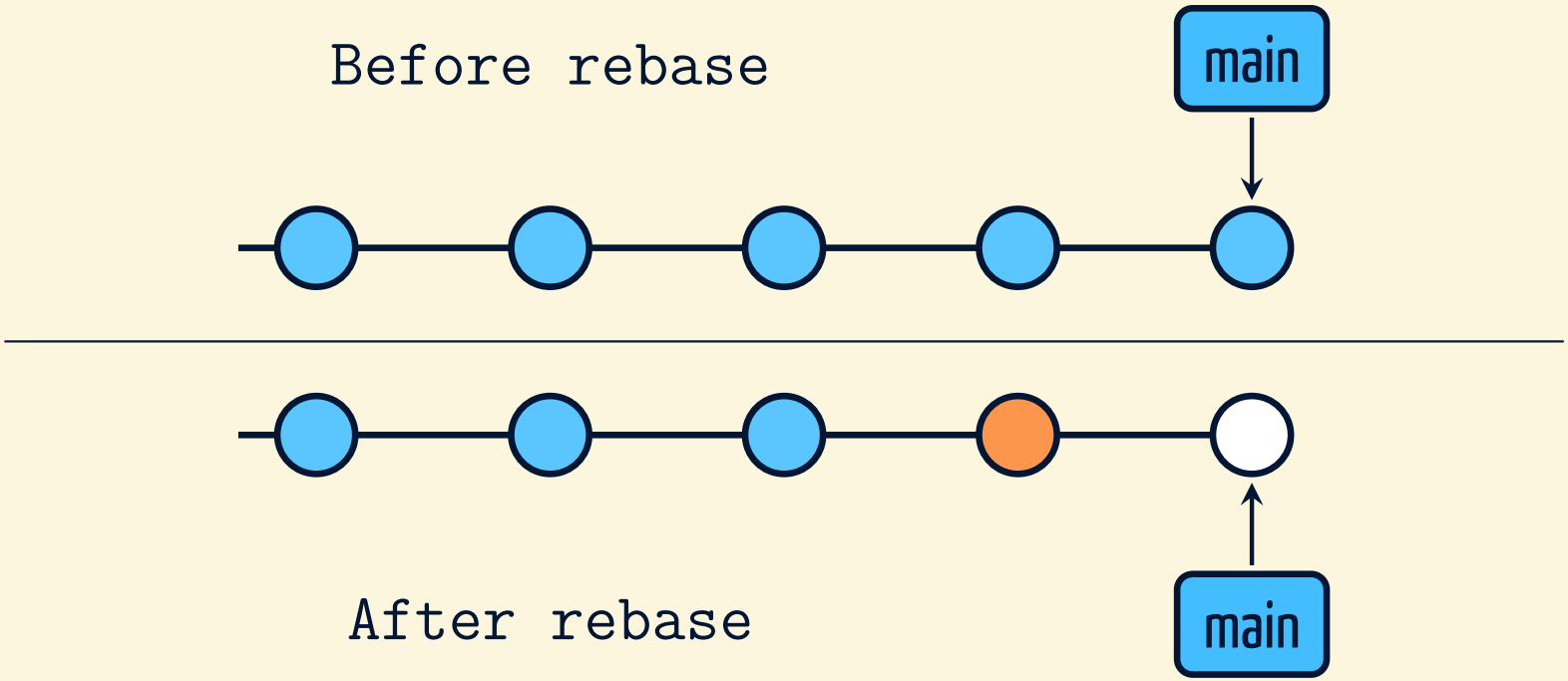
Which is the abstract idea of a rebase?

1. Go back in history till a **given** point
2. Replay **chosen** commits in their order
   ↳ possibly taking action on applied commit as specified



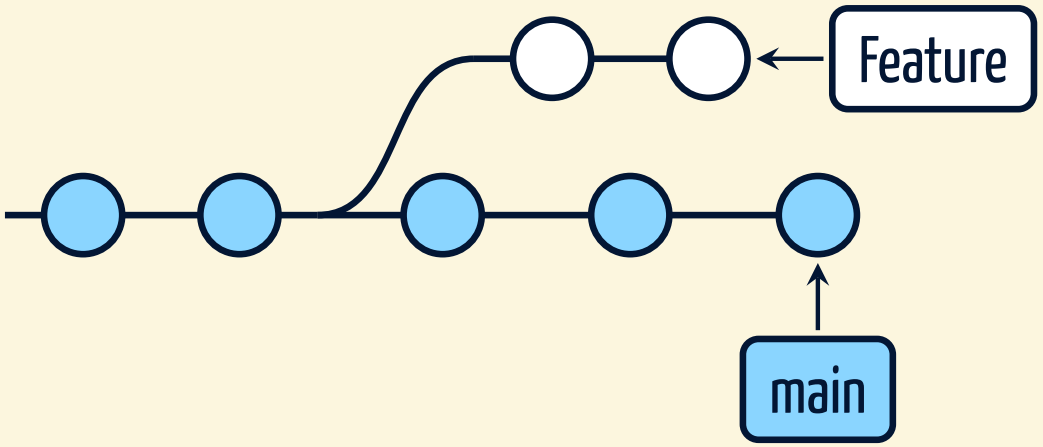
Which is the abstract idea of a rebase?

1. Go back in history till a **given** point
2. Replay **chosen** commits in their order
   ↳ possibly taking action on applied commit as specified

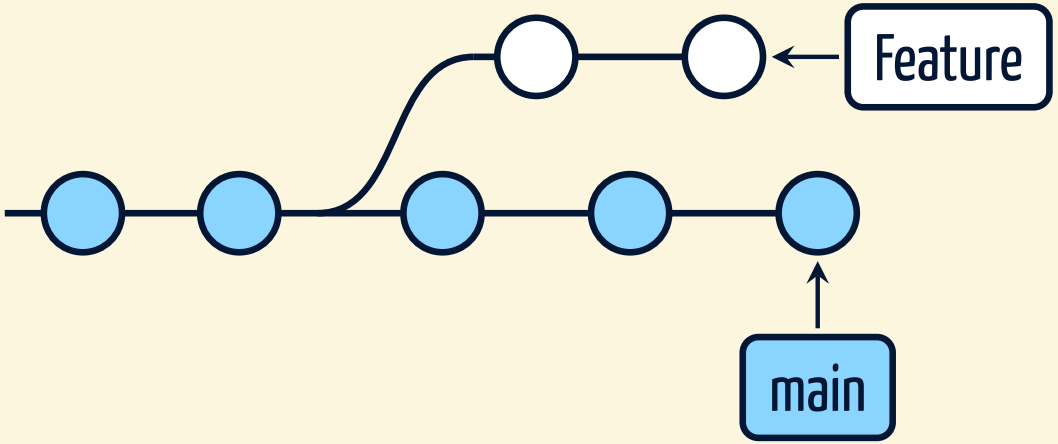


Rebasing instead of (three-way) merging

Three way merge from Feature:

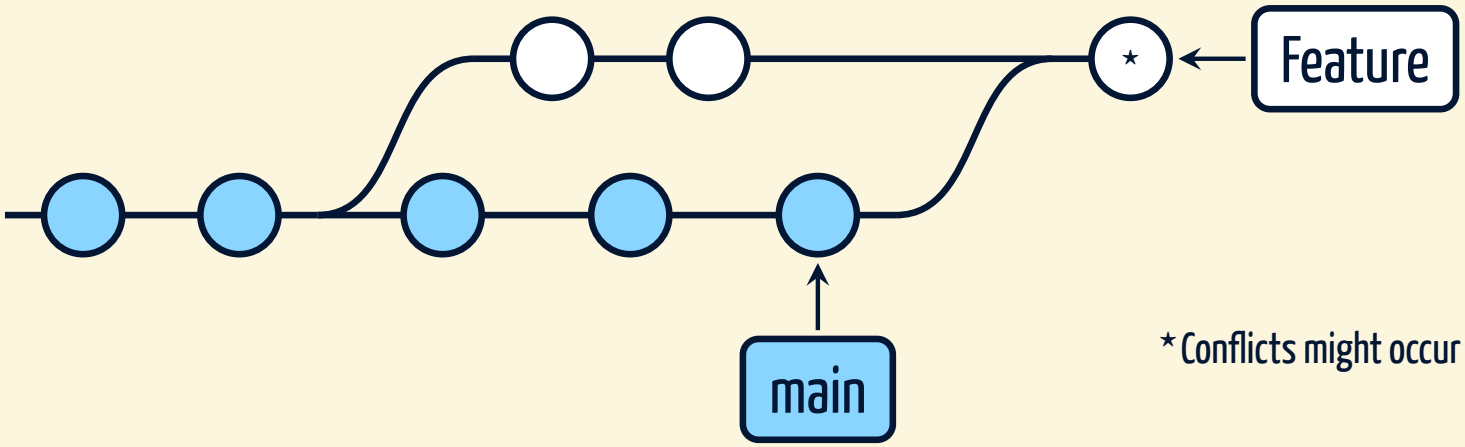


Rebase from Feature:



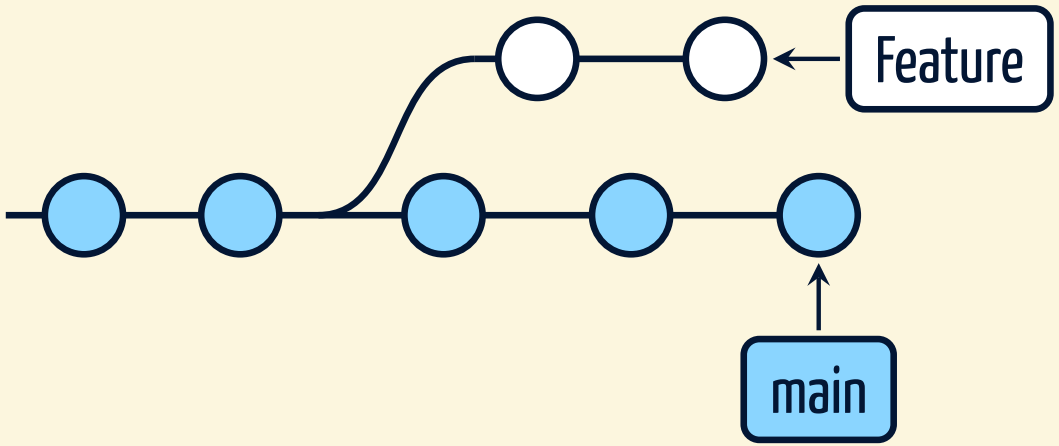
Rebasing instead of (three-way) merging

Three way merge from Feature:



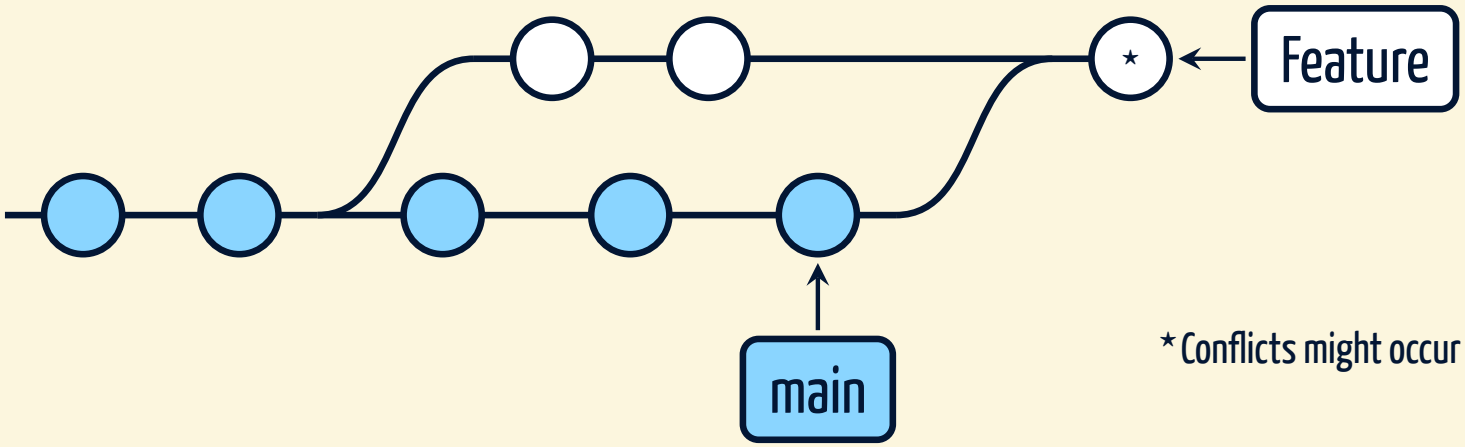
* Conflicts might occur

Rebase from Feature:



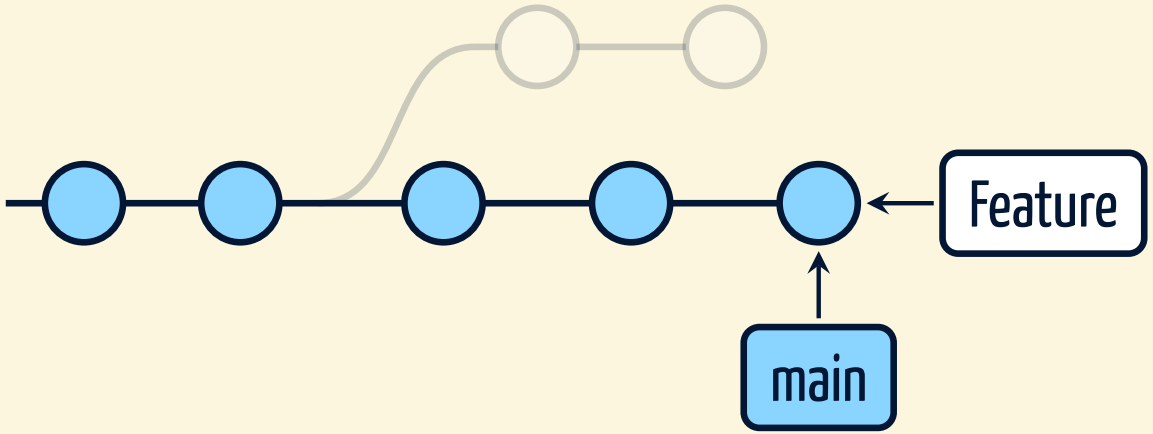
Rebasing instead of (three-way) merging

Three way merge from Feature:



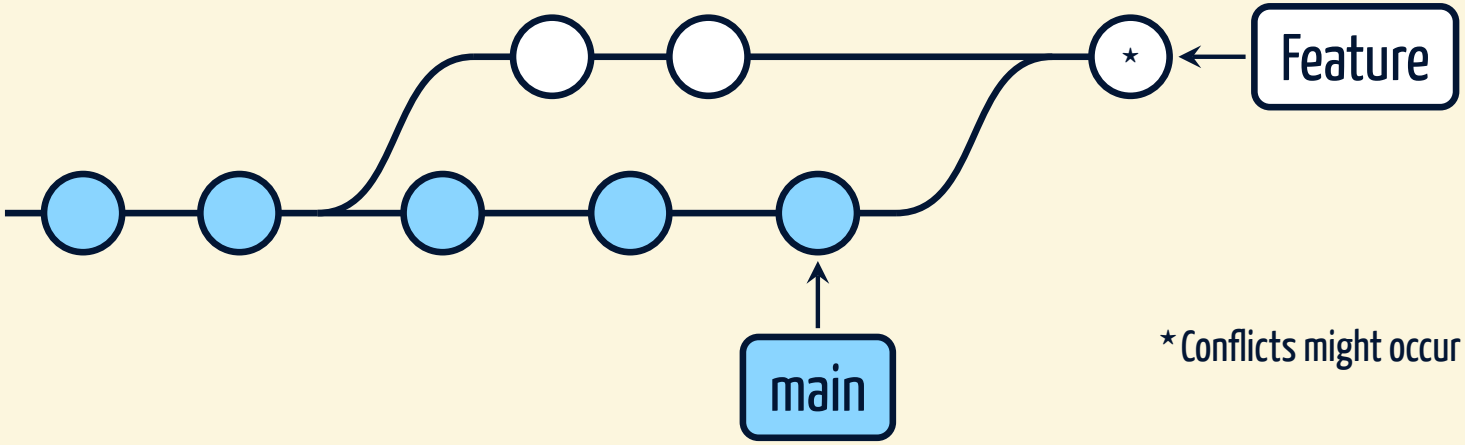
* Conflicts might occur

Rebase from Feature:



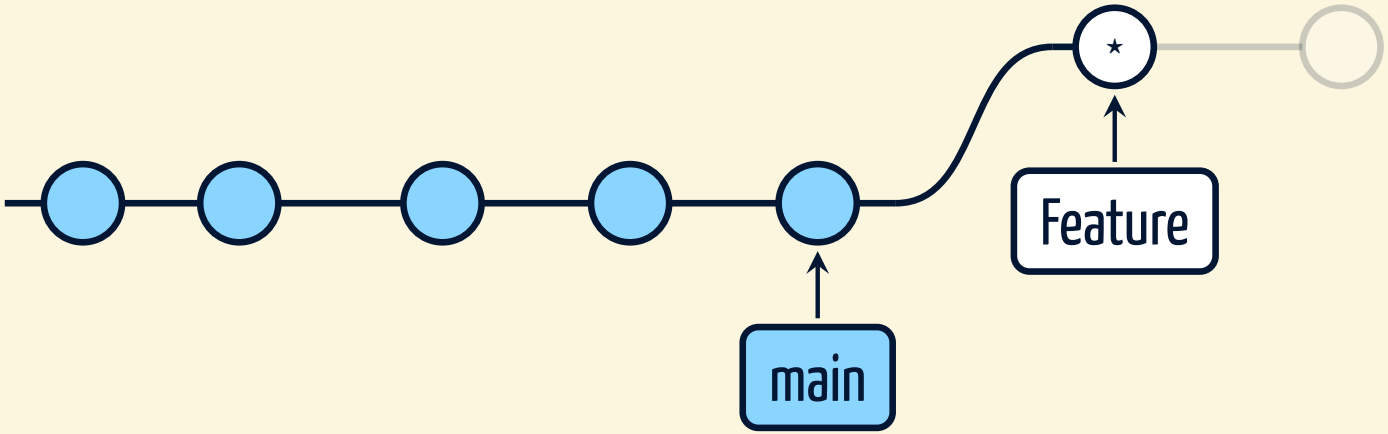
Rebasing instead of (three-way) merging

Three way merge from Feature:



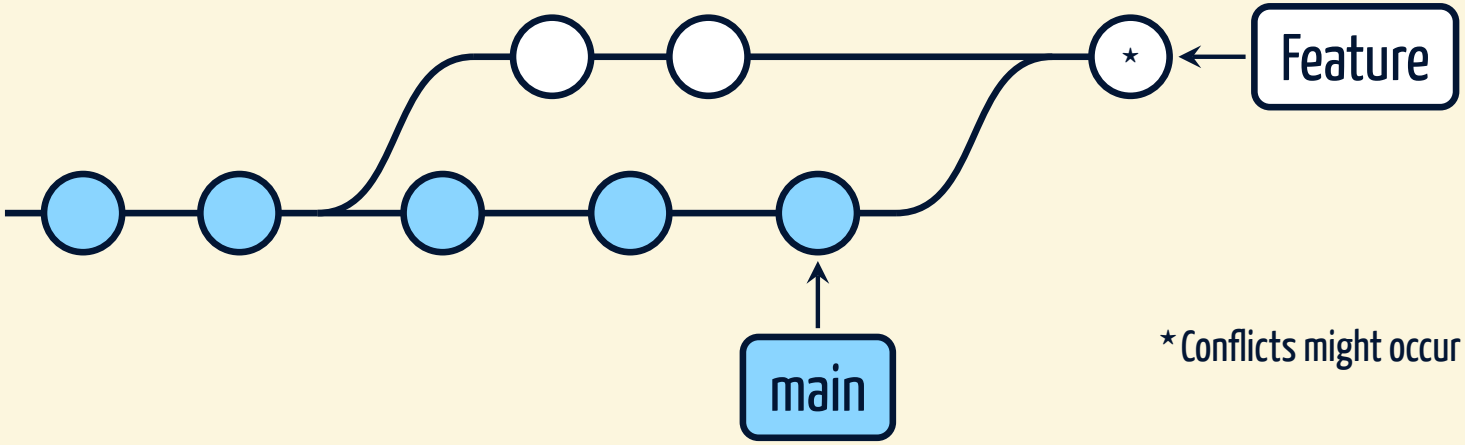
* Conflicts might occur

Rebase from Feature:



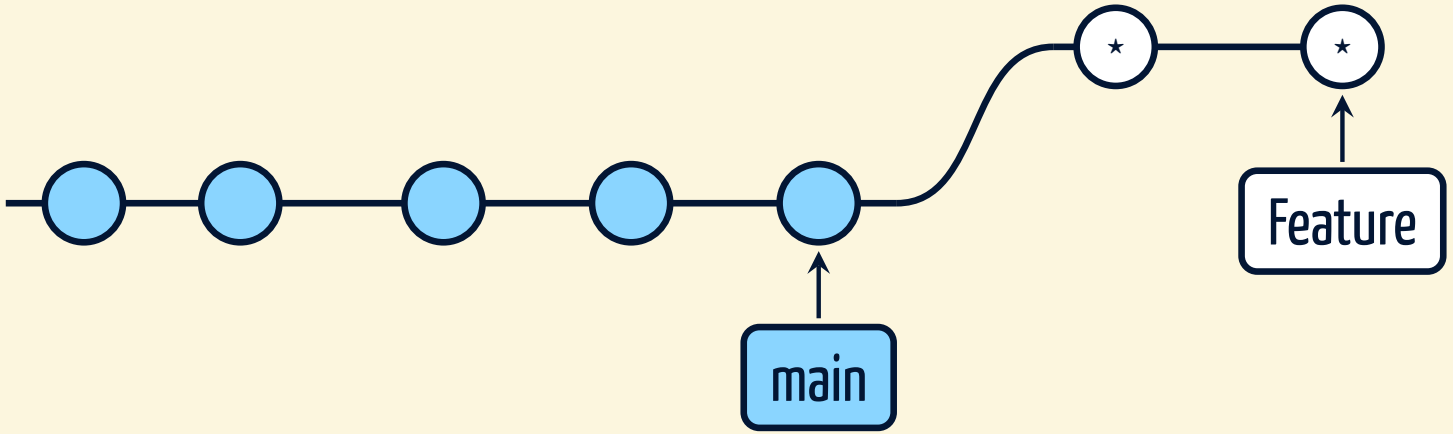
Rebasing instead of (three-way) merging

Three way merge from Feature:



* Conflicts might occur

Rebase from Feature:



Git rebase in its (almost) full glory

```
git rebase [-i] [--onto <newbase>] [<upstream> [<branch>]]
```

`-i` Specify it to set up the rebase interactively

`<newbase>` Where to apply the chosen commit
↳ by default `<upstream>`

`<upstream>` Base history for the rebase to choose commits

`<branch>` The branch from which the rebase is done

Conflicts might occur

1. Resolve them as usual and `git-add` the files
2. Run `git rebase --continue`

You can actually do more than that; see the official documenation to know more about rebasing.

Git rebase in its (almost) full glory

```
git rebase [-i] [--onto <newbase>] [<upstream> [<branch>]]
```

`-i`	Specify it to set up the rebase interactively
`<newbase>`	Where to apply the chosen commit ↳ by default `<upstream>`
`<upstream>`	Base history for the rebase to choose commits
`<branch>`	The branch from which the rebase is done

Conflicts might occur

- Run `git rebase --skip` to ignore commit
- Run `git rebase --abort` to end rebase

You can actually do more than that; see the official documenation to know more about rebasing.

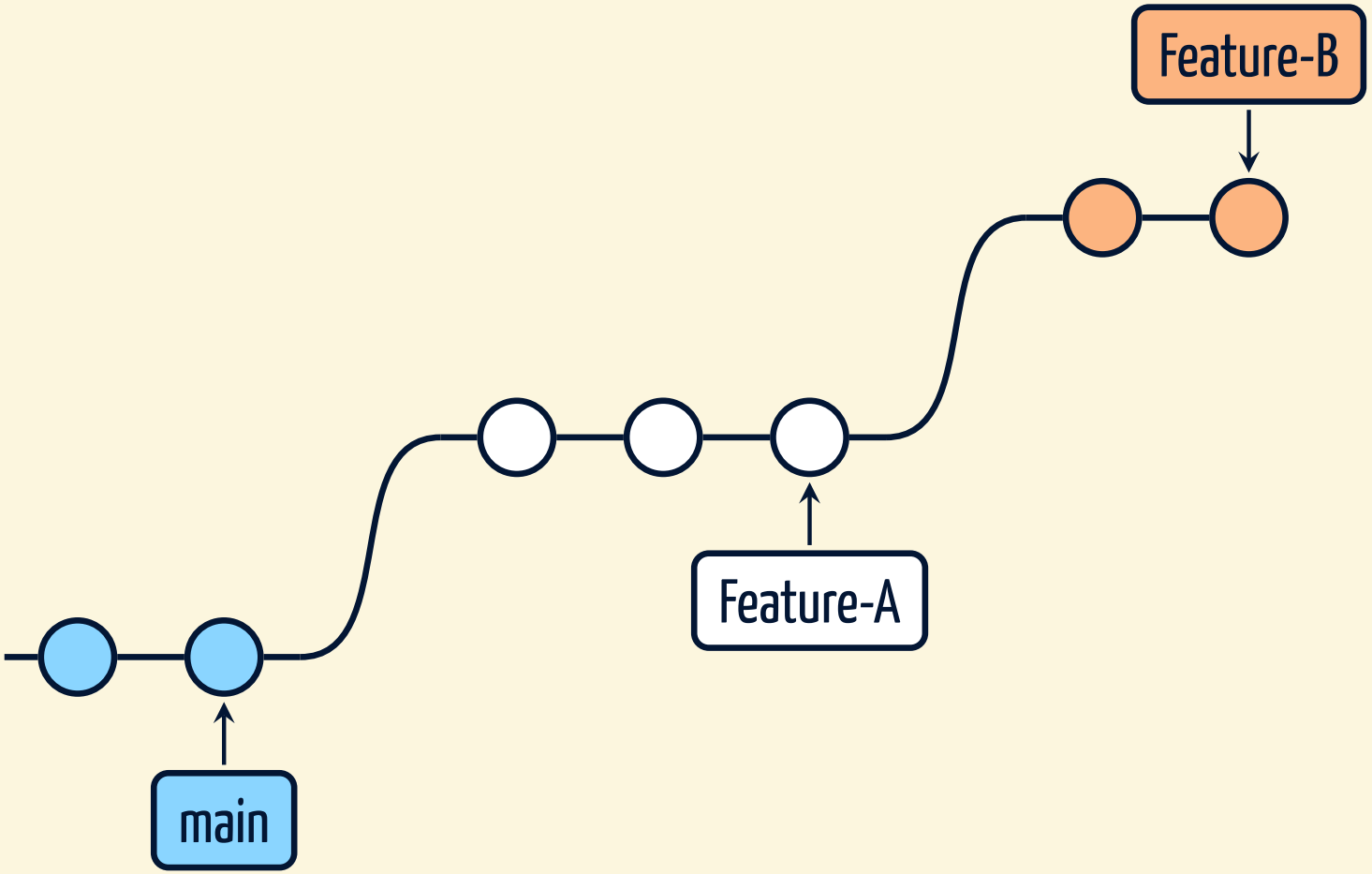
Git rebase in its (almost) full glory

```
git rebase [-i] [--onto <newbase>] [<upstream> [<branch>]]
```

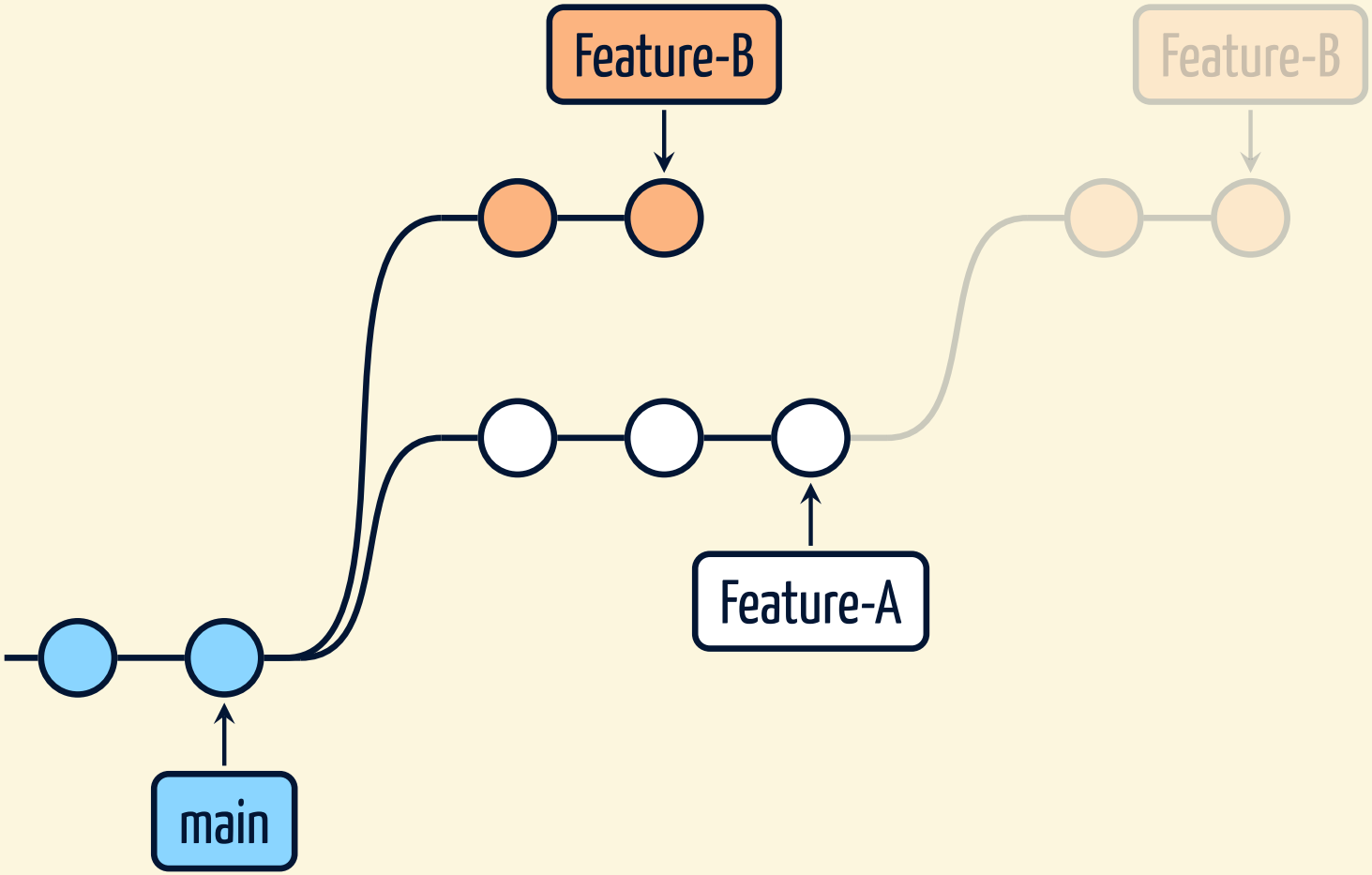
1. If `<branch>` is specified, git will switch to it
2. Commits in the current branch but that are not in `<upstream>` are saved to a temporary area
   ↳ Use `git log <upstream>..HEAD` to see these commits
3. The current branch is reset to `<upstream>`
   ↳ or `<newbase>` if the `--onto` option was supplied
4. The previously saved commits are then reapplied to the current branch, one by one, in order → conflicts might occur!
   ↳ already existing commits are by default omitted

You can actually do more than that; see the official documenation to know more about rebasing.

Another example



Another example




```
git rebase --onto main Feature-A Feature-B
```

Back to rebasing instead of merging

Useful to keep history clean in repository

If working **alone** on a branch

1. Get your work done
2. `git rebase main your-branch`
3. Then `git merge main` will be up-to-date
4. Switch to `main` and do a trivial merge

Interactive rebase

- It allows to act on commits while re-applying them
- It offers further possibilities to tidy up work

An example of interactive rebase

Adjust history of recent local changes

```
$ git log --oneline -n 4
e499d89 Deploy engine turbo           # This should be rephrased
f8b0c25 Improve flaps of wings
dfb705b Make some wheels maintenance # Here we forgot a file
a0a3f28 Work on cockpit instruments
```

An example of interactive rebase

Adjust history of recent local changes

```
$ git log --oneline -n 4
e499d89 Deploy engine turbo          # This should be rephrased
f8b0c25 Improve flaps of wings
dfb705b Make some wheels maintenance # Here we forgot a file
a0a3f28 Work on cockpit instruments
$ git add wheel_test.cpp
$ git commit -m "This commit message does not matter"
[airplane 364ff12] This commit message does not matter
Date: Mon Nov 28 11:57:01 2022 +0100
1 files changed, 546 insertions(+), 810 deletions(-)
```

An example of interactive rebase

Adjust history of recent local changes

```
$ git log --oneline -n 4
e499d89 Deploy engine turbo          # This should be rephrased
f8b0c25 Improve flaps of wings
dfb705b Make some wheels maintenance # Here we forgot a file
a0a3f28 Work on cockpit instruments
$ git add wheel_test.cpp
$ git commit -m "This commit message does not matter"
[airplane 364ff12] This commit message does not matter
Date: Mon Nov 28 11:57:01 2022 +0100
1 files changed, 546 insertions(+), 810 deletions(-)
$ git rebase -i HEAD~5  # --> Edit, save, exit from editor
```

```
pick a0a3f28 Work on cockpit instruments                #1
pick dfb705b Make some wheels maintenance               #2
pick f8b0c25 Improve flaps of wings                     #3
pick e499d89 Deploy engine turbo                        #4
pick 364ff12 This commit message does not matter        #5
# Rebase 9fdb3bd..e499d89 onto 9fdb3bd
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
# [...]
```

```
pick a0a3f28 Work on cockpit instruments                #1
pick dfb705b Make some wheels maintenance               #2
fixup 364ff12 This commit message does not matter       #5
pick f8b0c25 Improve flaps of wings                     #3
pick e499d89 Deploy engine turbo                        #4
# Rebase 9fdb3bd..e499d89 onto 9fdb3bd
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
# [...]
```

An example of interactive rebase

Adjust history of recent local changes

```
$ git log --oneline -n 4
e499d89 Deploy engine turbo          # This should be rephrased
f8b0c25 Improve flaps of wings
dfb705b Make some wheels maintenance # Here we forgot a file
a0a3f28 Work on cockpit instruments

$ git add wheel_test.cpp
$ git commit -m "This commit message does not matter"
[airplane 364ff12] This commit message does not matter
Date: Mon Nov 28 11:57:01 2022 +0100
1 files changed, 546 insertions(+), 810 deletions(-)

$ git rebase -i HEAD~5  # --> Edit, save, exit from editor

# When asked for, rephrase commit as wished
Successfully rebased and updated refs/heads/airplane.
```

An example of interactive rebase

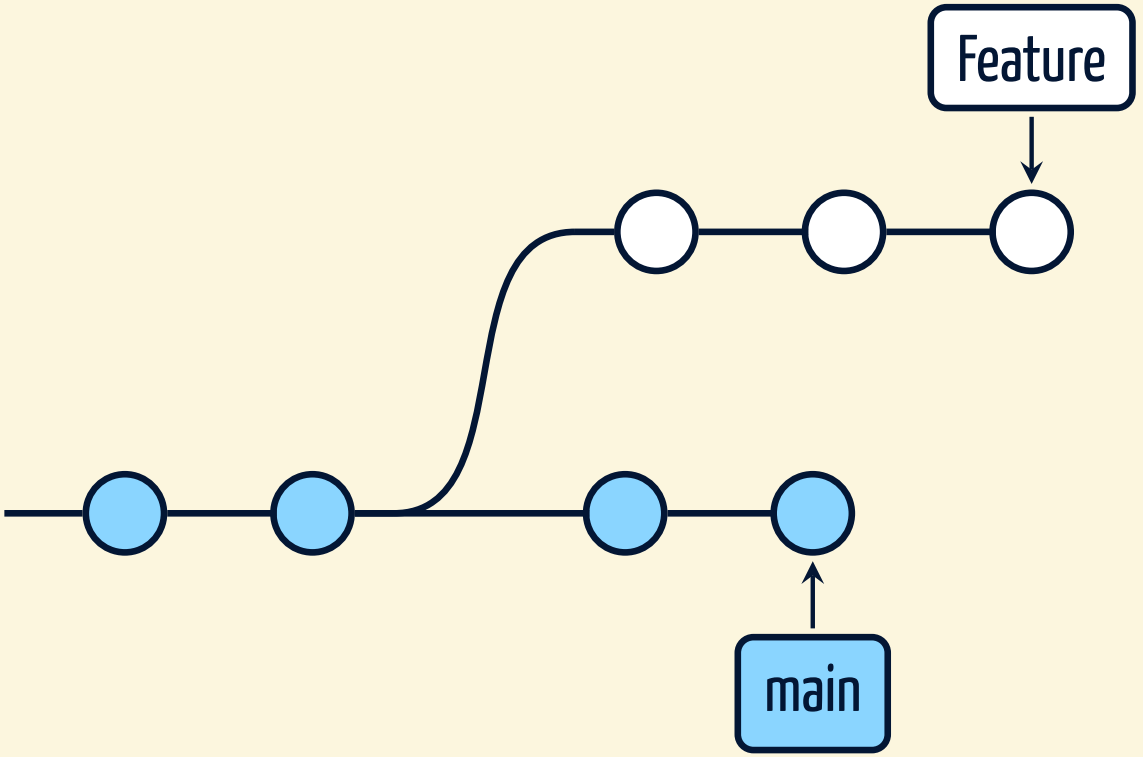
Adjust history of recent local changes

```
$ git log --oneline -n 4
e499d89 Deploy engine turbo          # This should be rephrased
f8b0c25 Improve flaps of wings
dfb705b Make some wheels maintenance # Here we forgot a file
a0a3f28 Work on cockpit instruments

$ git add wheel_test.cpp
$ git commit -m "This commit message does not matter"
[airplane 364ff12] This commit message does not matter
Date: Mon Nov 28 11:57:01 2022 +0100
1 files changed, 546 insertions(+), 810 deletions(-)

$ git rebase -i HEAD~5  # --> Edit, save, exit from editor

# When asked for, rephrase commit as wished
Successfully rebased and updated refs/heads/airplane.

$ git log --oneline -n 4
51a4b9b Deploy engine turbo and new pipes
afc765a Improve flaps of wings
9927a77 Make some wheels maintenance
a0a3f28 Work on cockpit instruments
```

History changed!

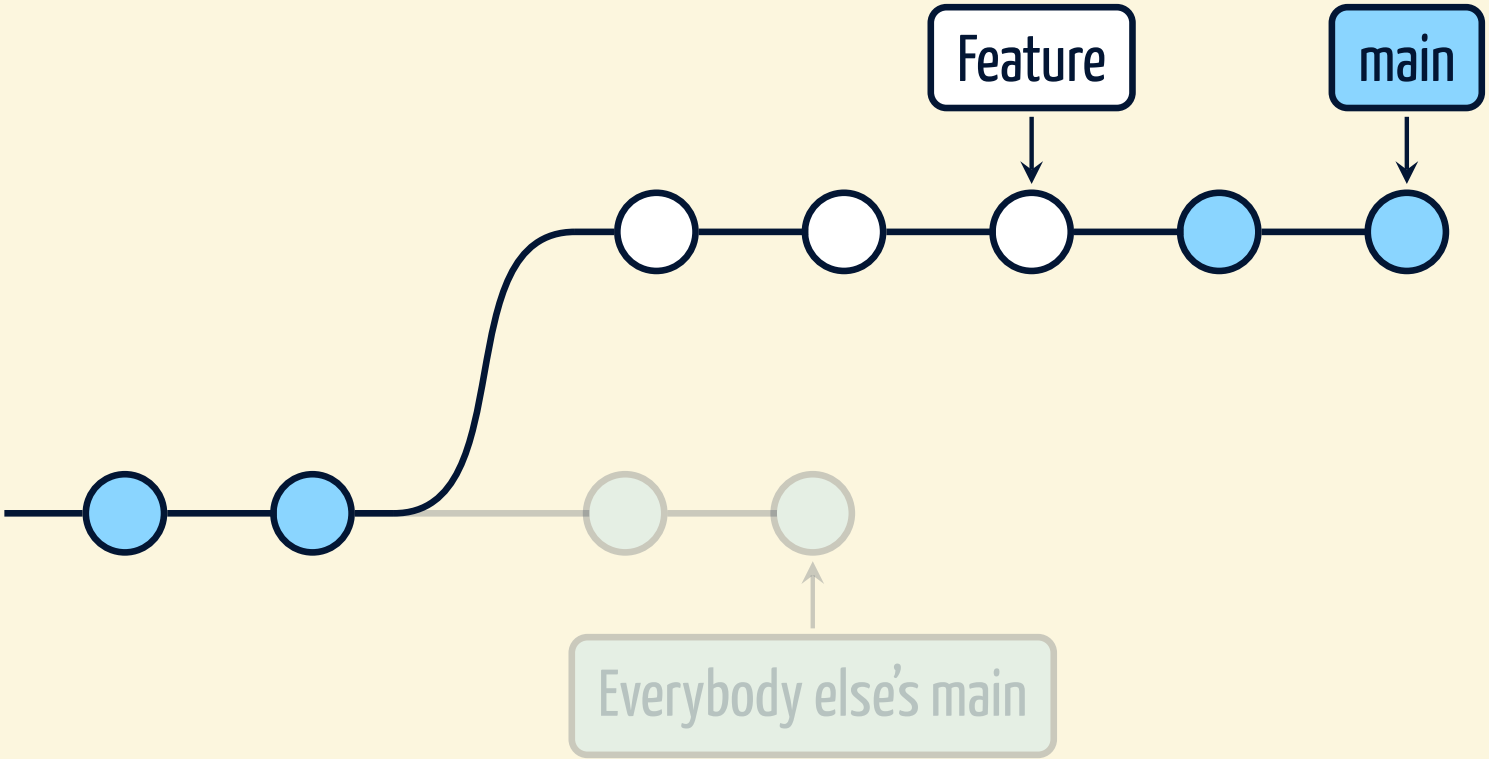
Coming back to changing public history

The golden rule of rebasing: **Never use it on public branches**



Coming back to changing public history

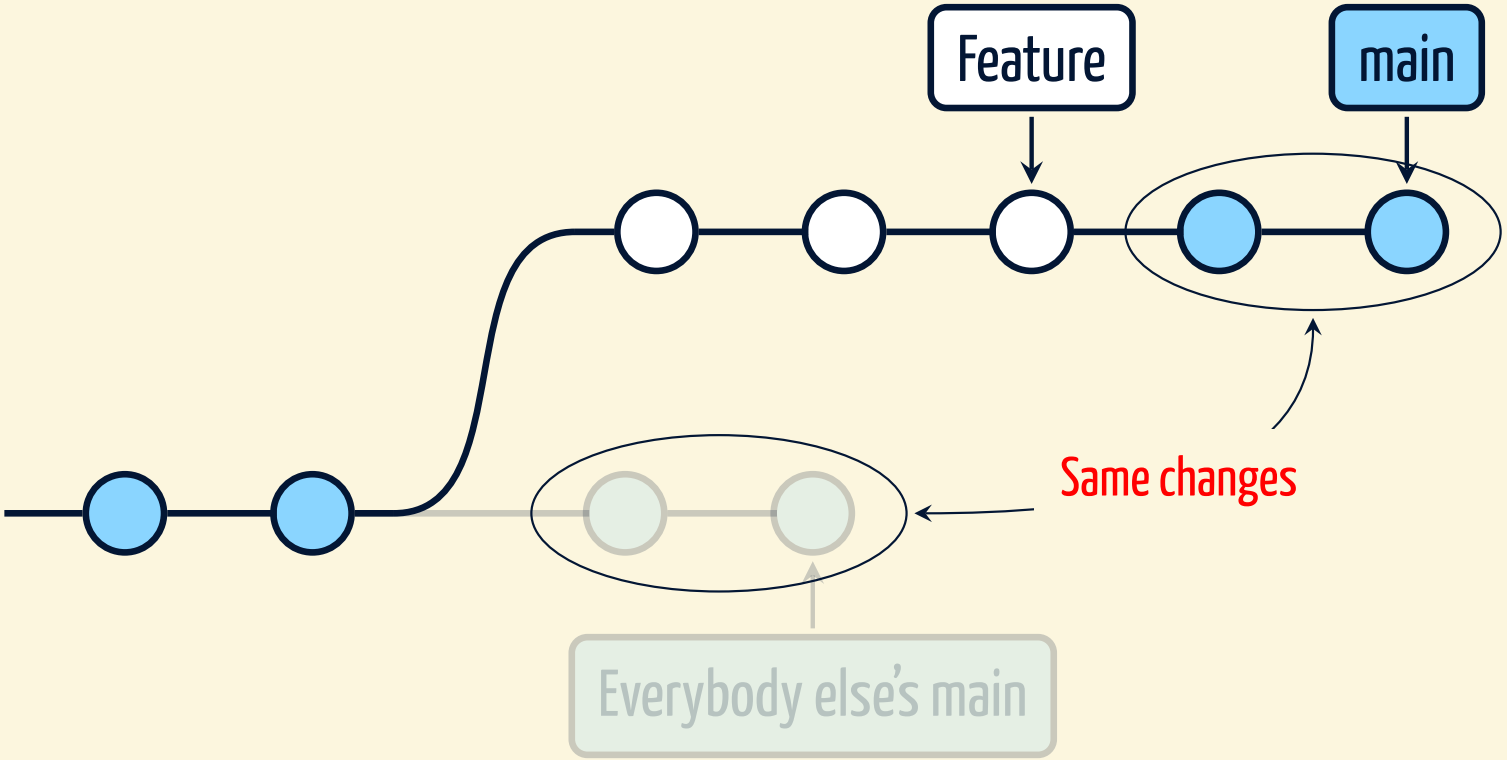
The golden rule of rebasing: **Never use it on public branches**




```
git rebase feature main # ...rebasing on main, AAAAAAAAARGH!
```

Coming back to changing public history

The golden rule of rebasing: **Never use it on public branches**




```
git rebase feature main # ...rebasing on main, AAAAAAAAARGH!
```

Git tag

Tagging the history of a repository

Lightweight tags

They simply are a name for an object (often a commit) and they are usually meant for private or temporary use

```
$ git tag v1.0
$ git tag
v1.0
$ git show v1.0
commit aa879f463acd41fc38c7e96090cc1eea279304df
Author: Alessandro Sciarra <sciarra@itp.uni-frankfurt.de>
Date:   Wed Aug 16 14:01:08 2025 +0200

Coffee machine production ready

# Commit differences
```

Tagging the history of a repository

Lightweight tags

They simply are a name for an object (often a commit) and they are usually meant for private or temporary use

Annotated tags

Tag objects contain a creation date, the tagger name and e-mail, a tagging message, and an optional GnuPG signature

Tagging the history of a repository

```
$ git tag -a v1.0 -m\
>     "Coffee machine software ready for production"
$ git show v1.0
tag v1.0
Tagger: Alessandro Sciarra <sciarra@itp.uni-frankfurt.de>
Date:   Wed Aug 17 09:12:24 2025 +0200

Coffee machine software ready for production

commit aa879f463acd41fc38c7e96090cc1eea279304df
Author: Alessandro Sciarra <sciarra@itp.uni-frankfurt.de>
Date:   Wed Aug 16 14:01:08 2025 +0200

Coffee machine production ready

# Commit differences
```

Prefer annotated tags for releases!

Tagging a code

Why should I?

- The codebase is released
- The codebase will be released
- The codebase is private but shared with colleagues
- The codebase will be (maybe) inherited
- **The software is used to produce data**

A milestone in development

A tag in the git history is to some extent a commitment, but it is also a statement of invaluable help about the codebase!

Semantic versioning

How should tags be named?

However you do it, use a CHANGELOG file to list user-relevant changes!

How should tags be named? `[Prefix]X[.Y[.Z]]`

According to the Semantic Versioning, increase

MAJOR version `X` when you make incompatible API changes
MINOR version `Y` when you add functionality in a backwards-compatible manner
PATCH version `Z` when you make backwards-compatible bug fixes

However you do it, use a CHANGELOG file to list user-relevant changes!

How should tags be named? `[Prefix]X[.Y[.Z]]`

According to the Semantic Versioning , increase

MAJOR version `X` when you make incompatible API changes
MINOR version `Y` when you add functionality in a backwards-compatible manner
PATCH version `Z` when you make backwards-compatible bug fixes

Choose your alternative, but give yourself a rule, e.g. increase

MAJOR version `X` when you introduce new big features
MINOR version `Y` when you add minor functionality or do big refactoring
PATCH version `Z` when you make bug fixes (without large changes)

However you do it, use a CHANGELOG file to list user-relevant changes!

How should tags be named? `[Prefix]X[.Y[.Z]]`

According to the Semantic Versioning, increase

MAJOR version `X` when you make incompatible API changes
MINOR version `Y` when you add functionality in a backwards-compatible manner
PATCH version `Z` when you make backwards-compatible bug fixes

Choose your alternative, but give yourself a rule, e.g. increase

MAJOR version `X` when you introduce new big features
MINOR version `Y` when you add minor functionality or do big refactoring
PATCH version `Z` when you make bug fixes (without large changes)

MAJOR version `X` when you introduce new features
MINOR version `Y` when you do big refactoring or fix bugs

However you do it, use a CHANGELOG file to list user-relevant changes!

Ideally

```
$ git tag -n
  v2.0.0   Coffee machine ready for milk drinks      Major
  v1.3.1   Fix milk temperature problem              Patch
  v1.3.0   Interface milk system with machine        Minor
  v1.2.0   Add foam generator                        Minor
  v1.1.0   Add milk container                        Minor
  v1.0.0   Coffee machine production ready           Major
  v0.7.1   Fix pipe failures                         Patch
  v0.7.0   Interface all components                  Minor
  v0.6.0   Add coffee grounds container              Minor
  v0.5.0   Add water tray                            Minor
  v0.4.0   Add water tank                            Minor
  v0.3.0   Add bean container                        Minor
  v0.2.0   Add brew system and core engine           Minor
  v0.1.0   Deploy coffee machine skeleton            Minor
```

- Clear evolution of the codebase for everybody
- **v1.0** usually refers to the first stable version of the software

Summary so far

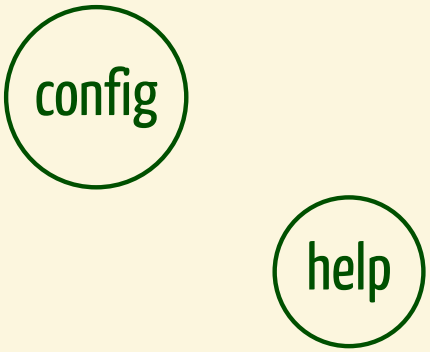
Gitflow

5 Gitflow

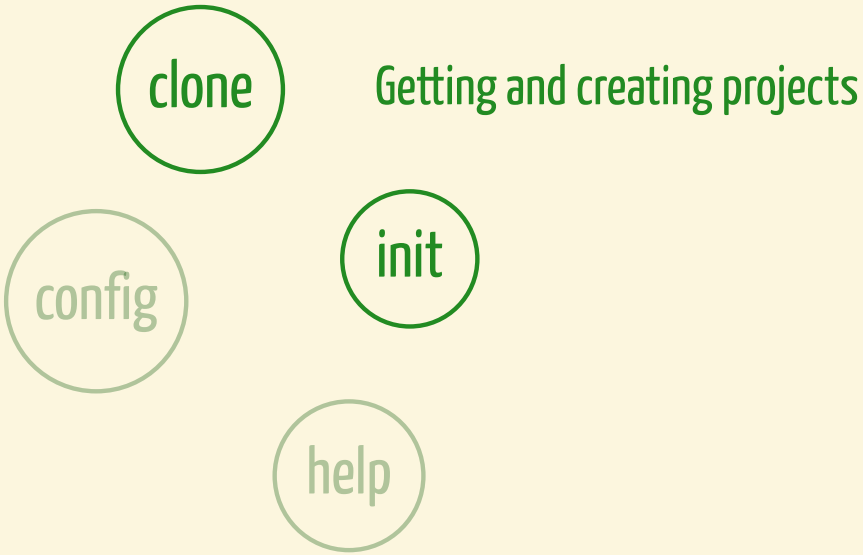
- Motivation and background
- The main branches
- Feature branches
- Release branches
- Hotfix branches
- Git-flow: an additional tool

Most used Git commands

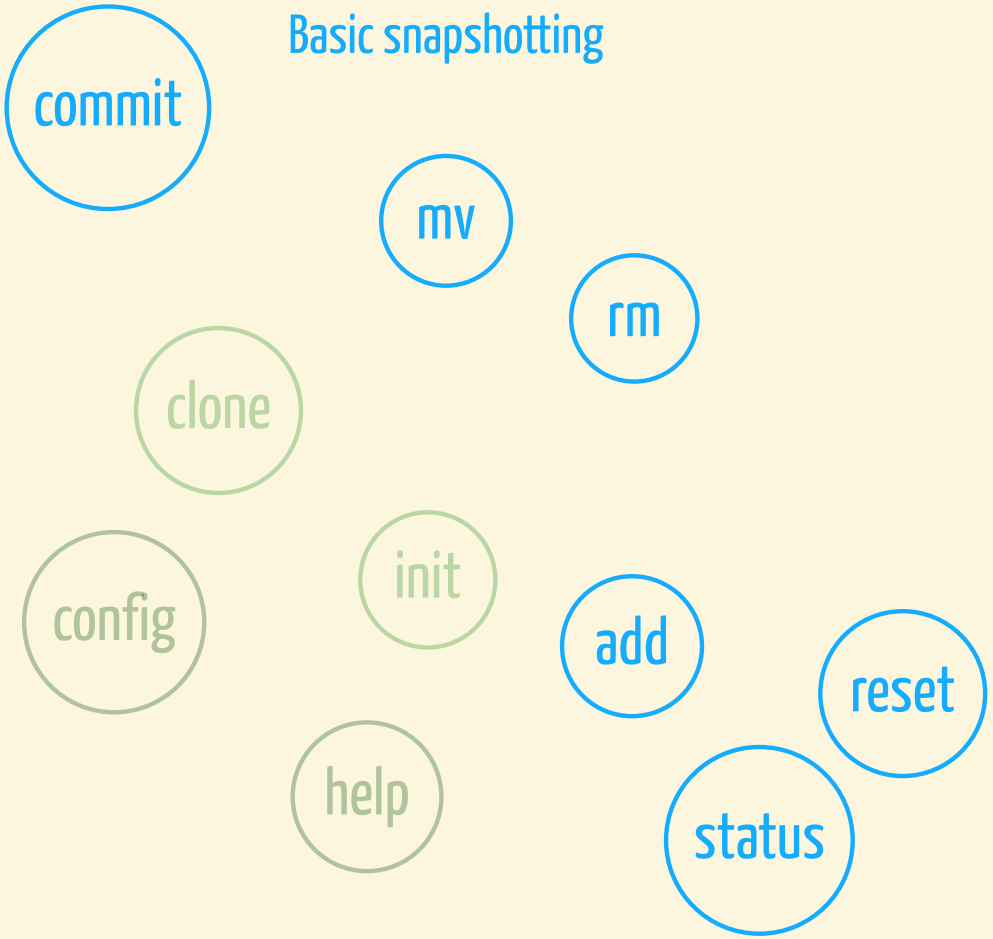
Setup and Config



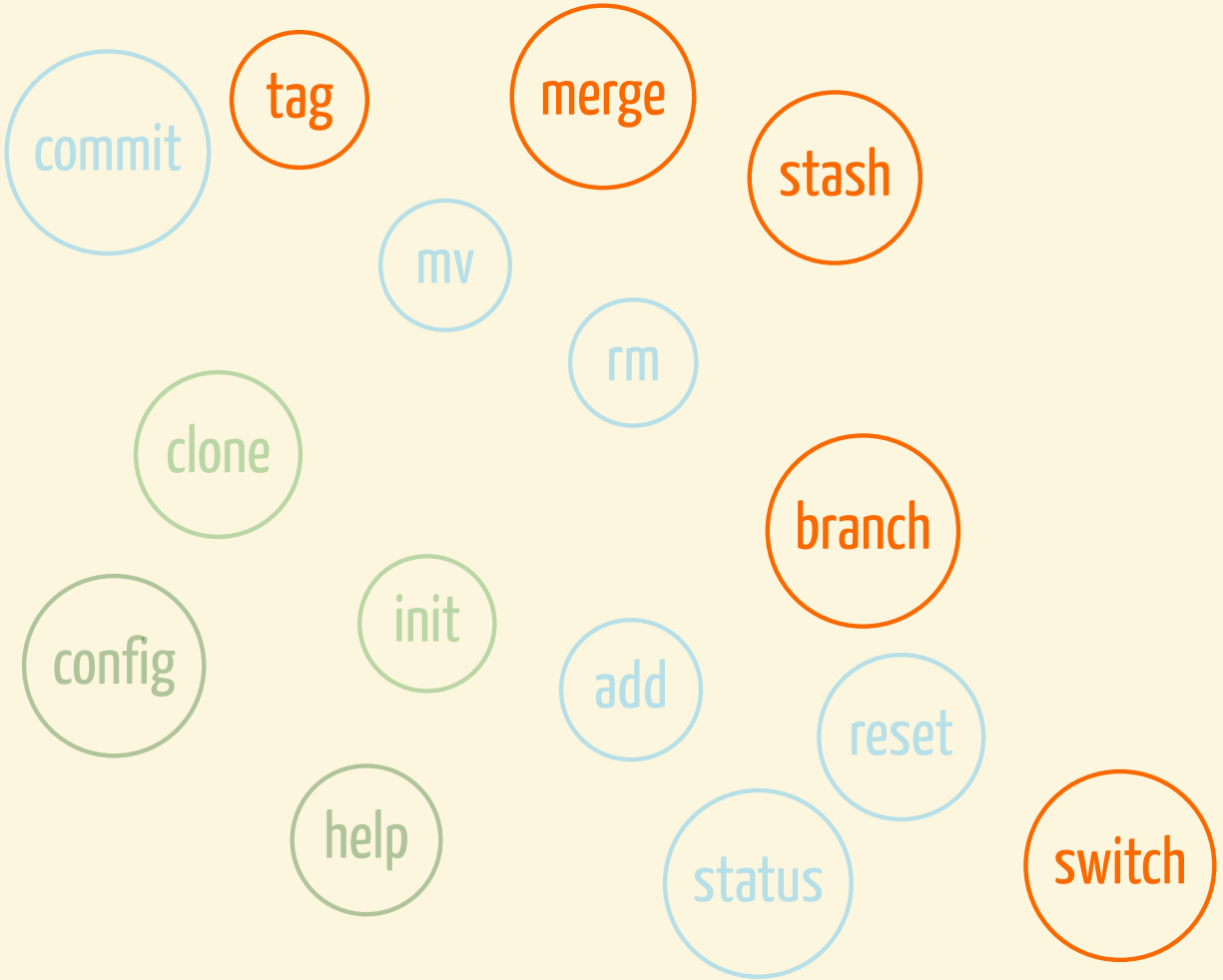
Most used Git commands



Most used Git commands

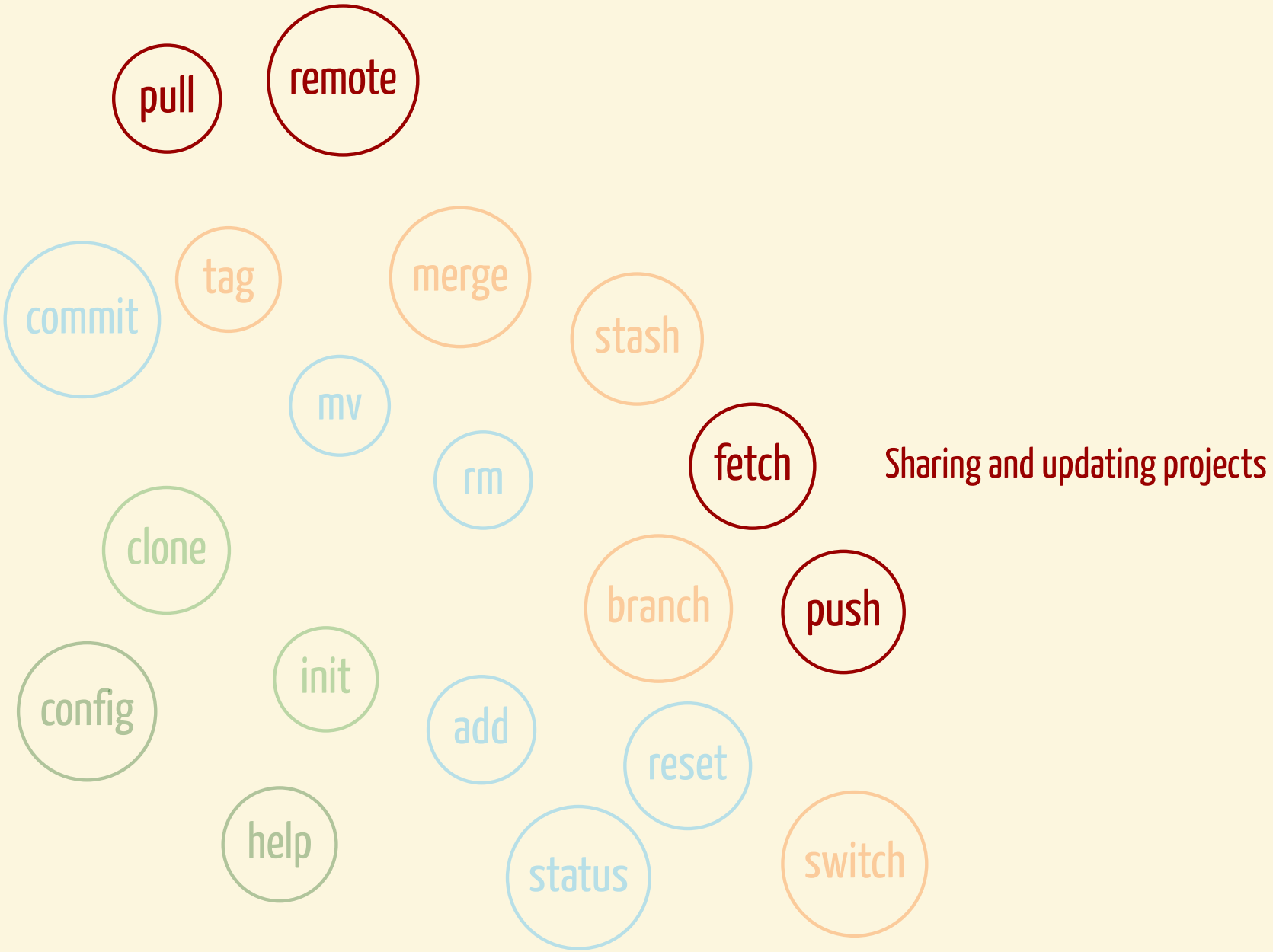


Most used Git commands

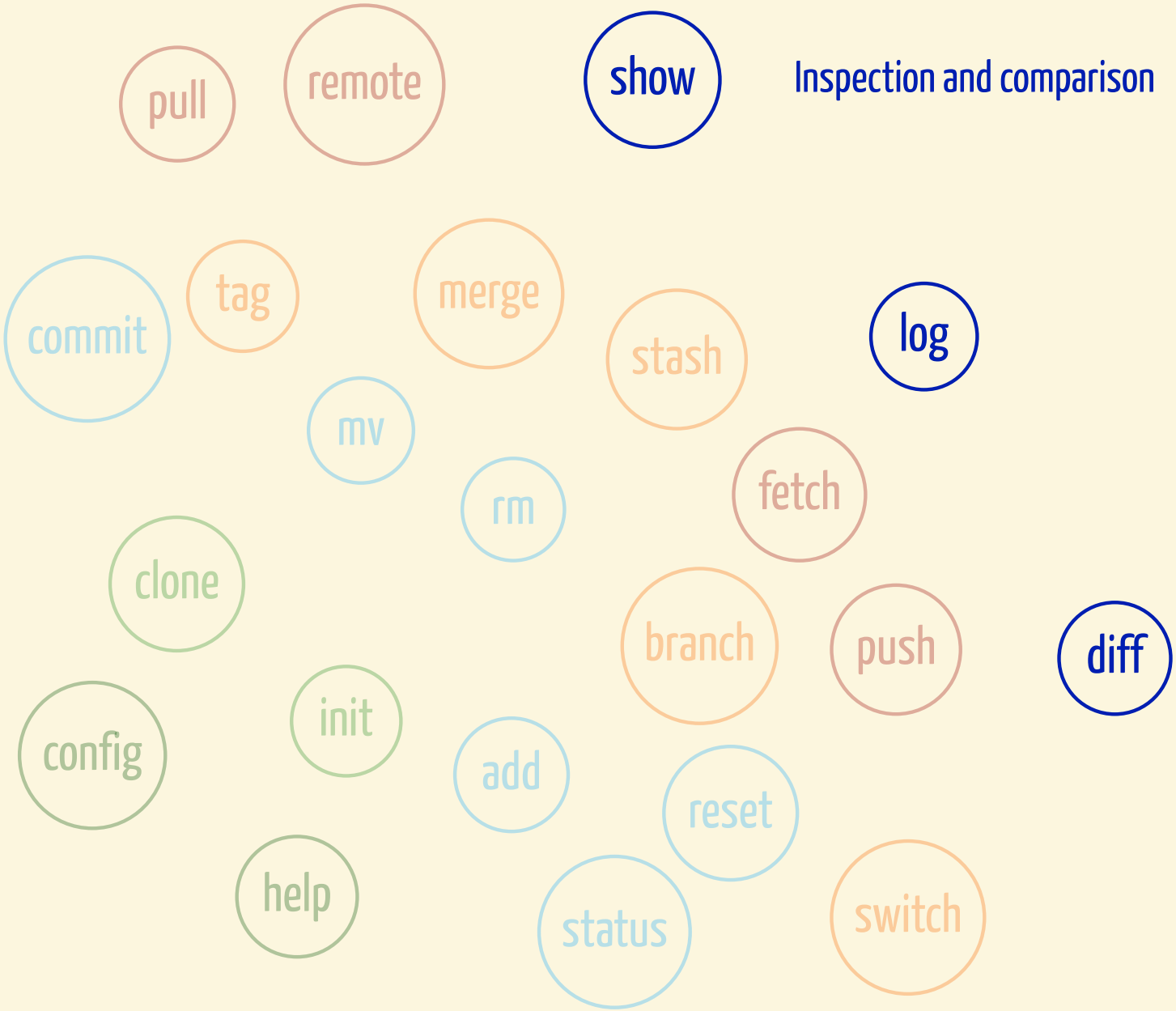


Branching and merging

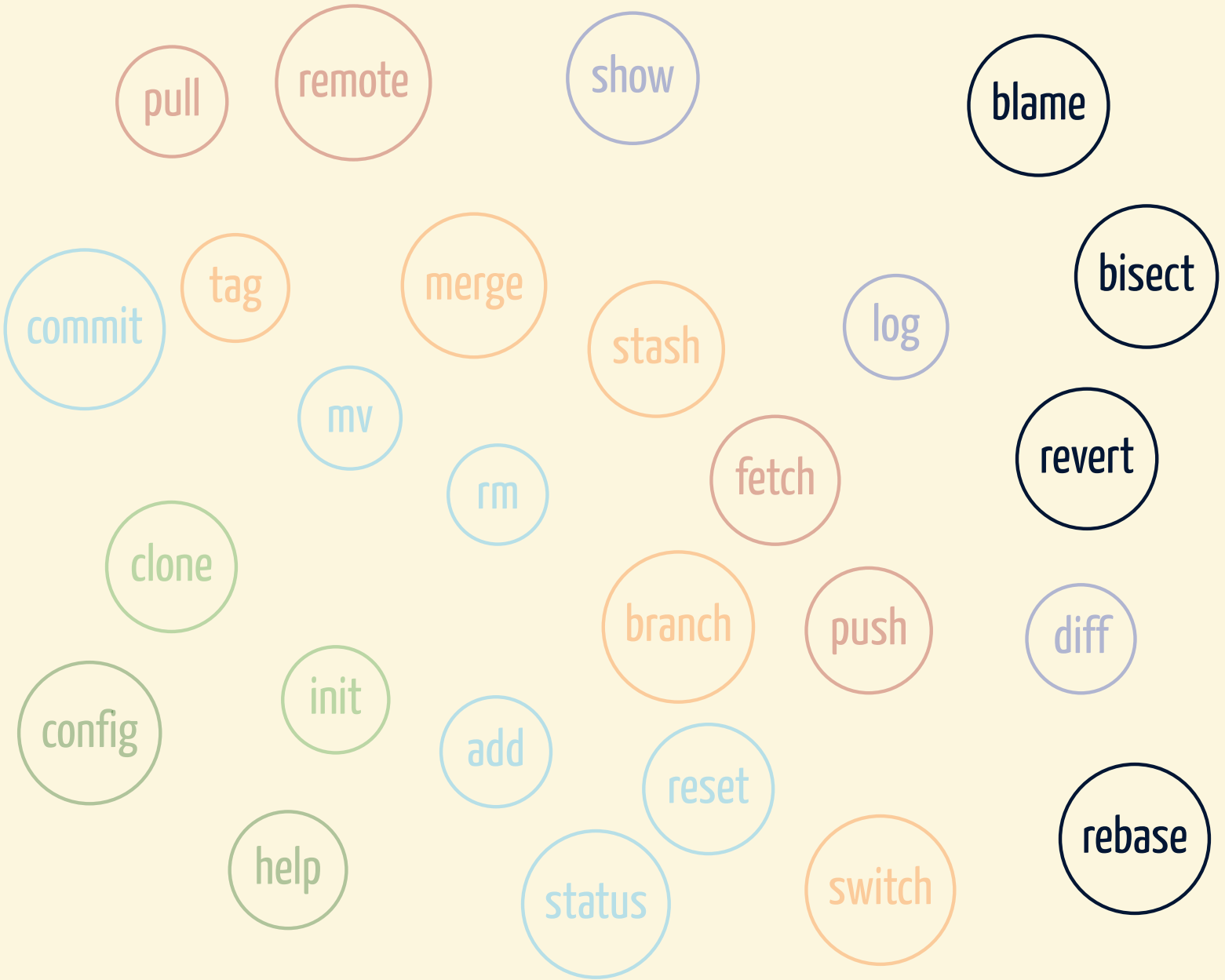
Most used Git commands



Most used Git commands

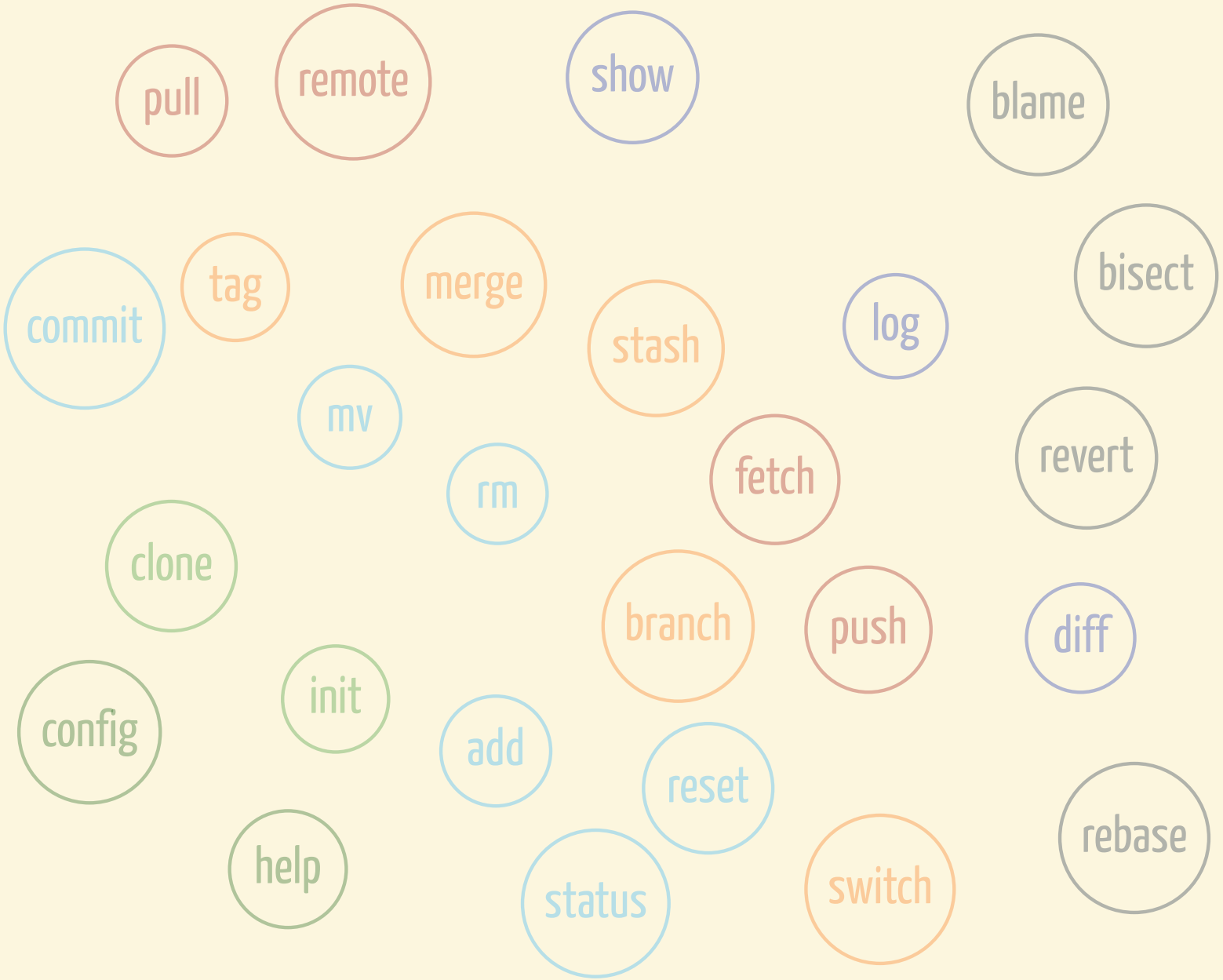


Most used Git commands



Most used Git commands

Git Documentation



Most used Git commands

Git Documentation

pull
remote
show
blame
tag
merge
bisect
commit
log
revert
fetch
clone
branch
push
diff
init
config
add
reset
rebase
help
status
switch