Git Branches

Outline

- Branches, merge and more `Amir`
  - creating your second branch 😉 (branch - checkout - switch)
  - deleting a branch
  - merge baby merge (git merge)
  - types of merge (FF - m)
  - rebase your history (git rebase)
  - rebase vs merge internal war

Movement Draft

1. *slides*: what is a branch why you should always branch
2. *code*: create a branch
   1. create repo
   2. commit "A" into README.md
   3. create branch foo
3. *search*: search for the branch in the .git folder
   1. why branches are nearly free
4. *code*: switch to the new branch
   1. commit "B" and "C"
5. *question*: what you do after you finish your best work? *you merge*
6. *slides*: 2 ways to handle this situation **merge** and **rebase*
   1. we will see first merge which has 2 possible outcomes
7. *code*: **MAKE A DIVERGENCE MERGE**
   1. switch to main
   2. commit "D" and "E" into main.md
   3. make a new branch here named main-merge-foo `(effectively main)`
   4. merge
   5. git log --graph --parents --oneline
   6. see the merge
8. *slides*: divergence merge, best common ancestor, merge commits
9. *code*: **MAKE A FAST_FORWARD MERGE**`
   1. switch to main
   2. make and checkout branch bar
   3. commit "X", "Y" into bar.md
   4. switch to main
   5. and merge bar
10. *slides*: **REBASE**

1. what is rebase
2. why some people hate it (altering history)
3. show current setup
4. what rebase will do *and what that enables us to do* - more in that in last point
    1. Explain the actual steps

11. *code*: make a rebase at `foo-rebase-main`
    1. checkout foo
    2. create and checkout foo-rebase-main
    3. rebase main
12. *slides*: pros and cods
    1. pros:
        - 👍 clean history
        - 👍 better in searching
    2. cons:
        - 👎 have to use `*force*` when pushing to remote because it alters branch history
13. **slides**: **Always merge on public branches, rebase | merge on private branches**

Working Graph changes:

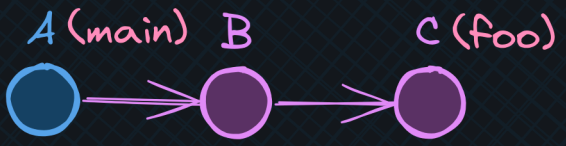
Start

A (main)

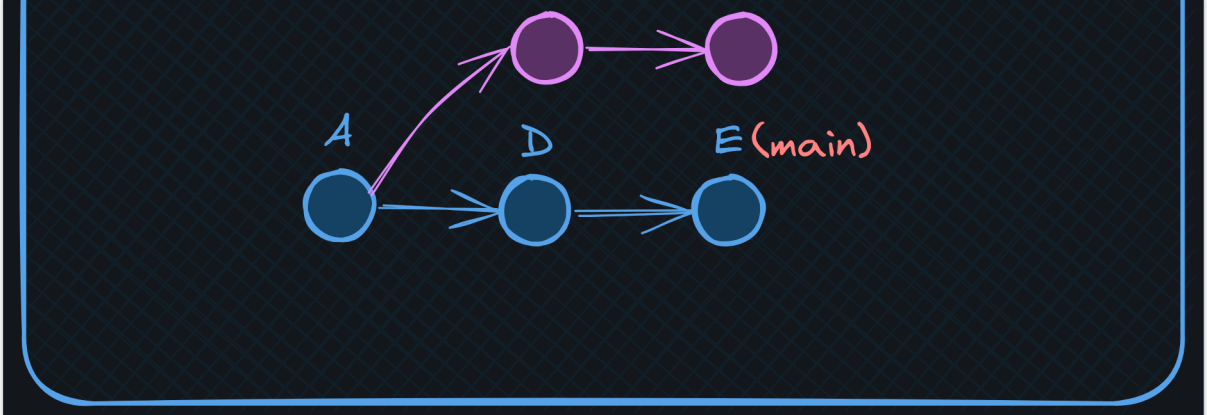
Create Branch Foo

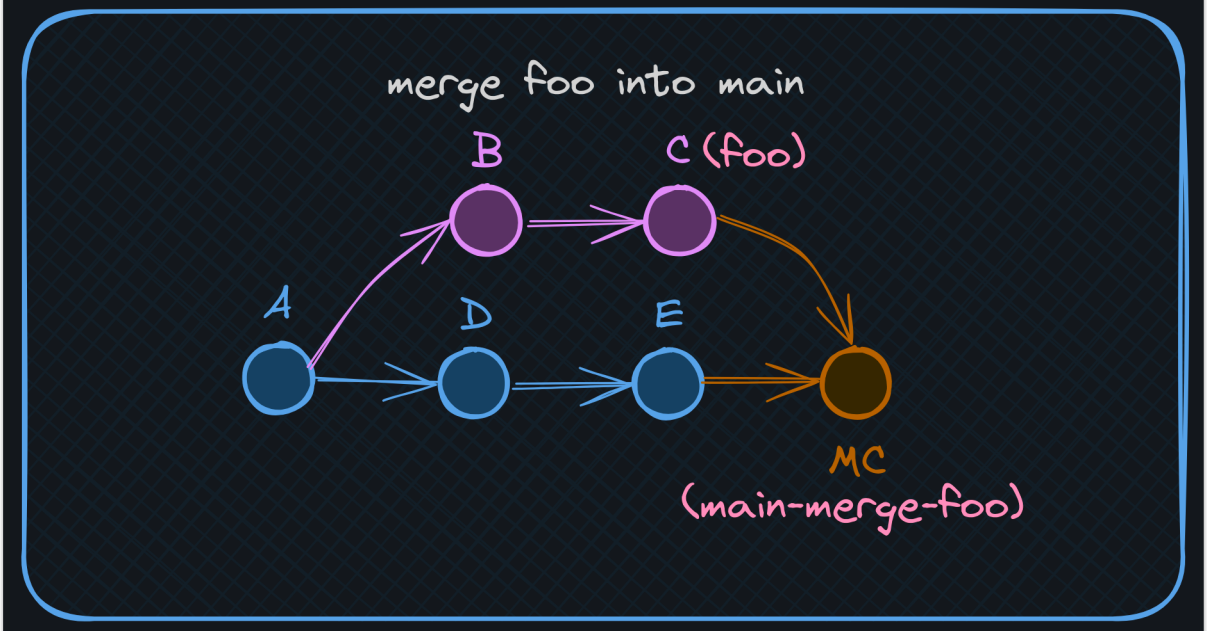
A (main, foo)

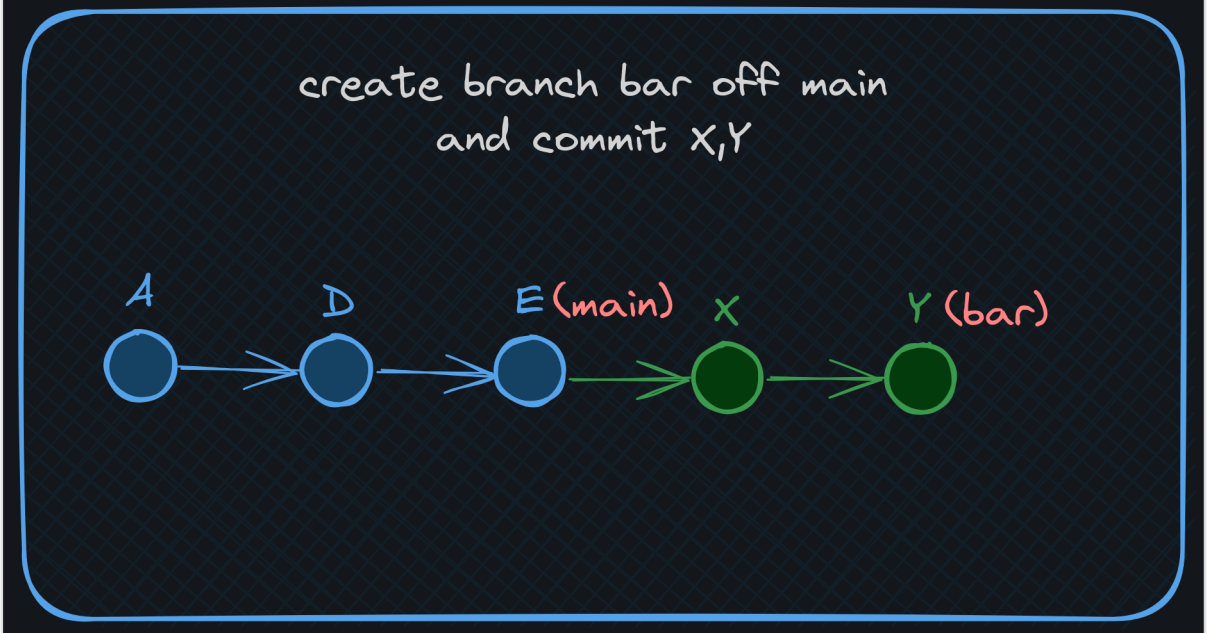
commit B,C into foo

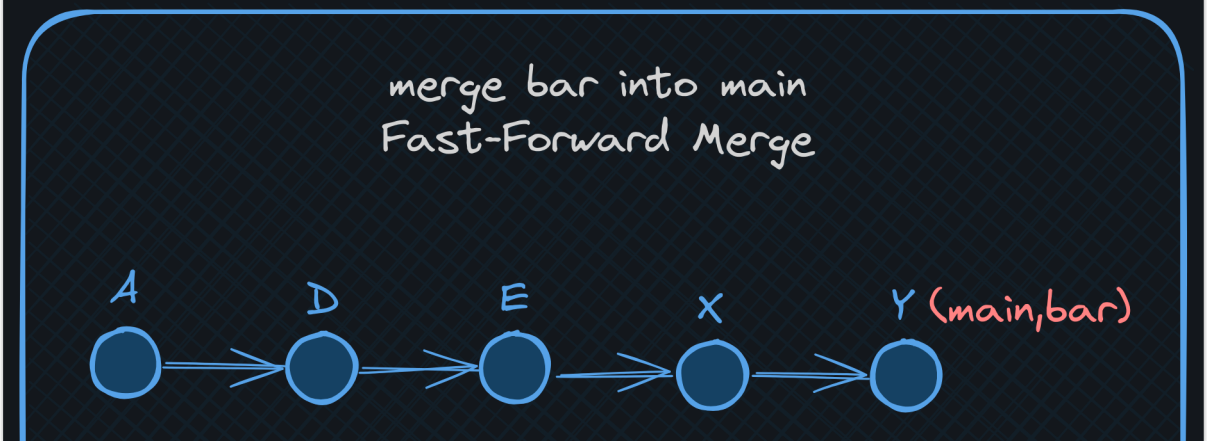


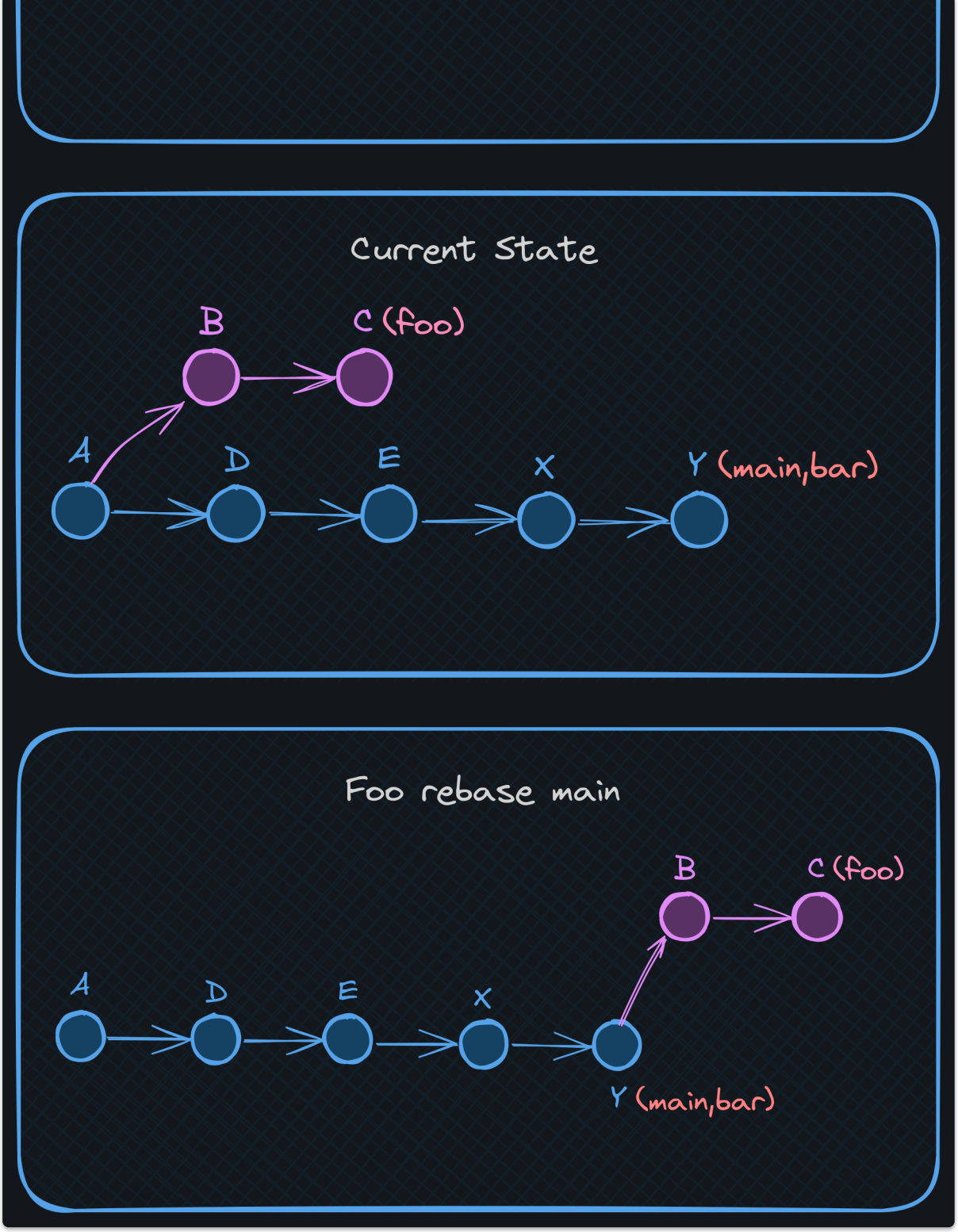
commit D,E into main


A
D
E (main)


merge foo into main
B
C (foo)
A
D
E
MC
(main-merge-foo)


create branch bar off main
and commit X,Y
A
D
E (main)
X
Y (bar)


merge bar into main
Fast-Forward Merge
A
D
E
X
Y (main,bar)



COMMANDS

```
# step 1 - create repo initial commit
mkdir git-branches
cd git-branches
git init
echo "A" >> README.md
git add .
git commit -m "A"
git log --graph --oneline

# step 2 - new branch
git branch foo
git checkout foo
find .git
```

```
# step 3 - 2 commit in foo
echo "B" >> README.md
git add .
git commit -m "B"
echo "C" >> README.md
git add .
git commit -m "C"
git log --graph --oneline

# step 4 - 2 commits in main
git checkout main
echo "D" >> second.md
git add .
git commit -m "D"
echo "E" >> second.md
git add .
git commit -m "E"
git log --graph --oneline

# step 5 - divergence merge
git checkout main
git checkout -b main-merge-foo
git merge foo
git log --graph --oneline --parents

# step 6 - prepare bar for fastforward merge
git checkout main
git checkout -b bar
echo "X" >> bar.md
git add .
git commit -m "X"
echo "Y" >> bar.md
git add .
git commit -m "Y"
git log --graph --oneline --parents

# step 7 - fastforward merge
git checkout main
git merge bar
git log --graph --oneline --parents

# step 8 - rebase
git checkout foo
git rebase main
git log --graph --oneline
```

Slides content:

1. what is a branch:

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

- branches are virtually free in git just a SHA to a commit/tree

2. what is a merge?

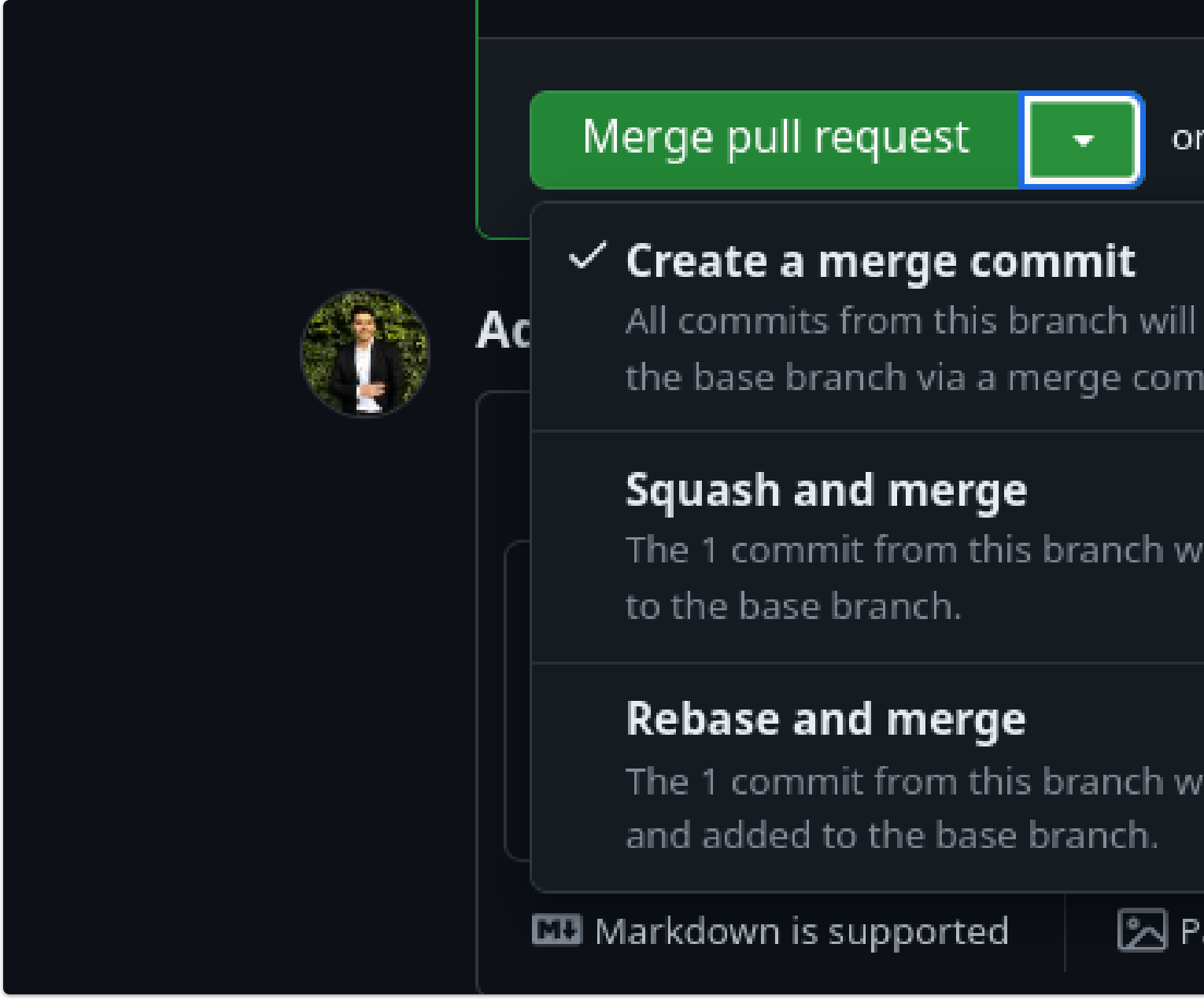
> ```
> A merge is attempting to combine two histories together that have diverged at some point in the past. There is a common commit point between the two, this is referred to as the best common ancestor
> ```

- merging have 2 different outcomes depending on the state of the history
  - **Fast Forward**: just update the pointer/reference (no merge commits)
  - **Divergence merge**: create a merge commit to combine 2 commits/histories have 2 parents

3. what is rebasing?

> *git-rebase - Reapply commits on top of another base tip - "the docs"*

4. How rebase works? executing `git rebase <targetbranch>` - `targetbranch` is often main
   1. checkout the latest commit at `<targetbranch>`
   2. play one commit at a time of the current branch - `current branch is often the feature branch`
   3. update current branch to the latest `SHA`
      - **be careful rebase alters history so don't ever rebase main/ public branches**
5. merge vs rebase:
   - **Merge**:
     - doesn't alter history
     - doesn't require `push force`
     - works with private and public branches without problems
     - makes annoying merge commits
   - **Rebase**:
     - alters history
     - requires `push force`
     - works with private branches only
     - no annoying merge commits
     - linear history which is easier to search
6. **workflows war**:
   1. Merge flow (just merge)
      1. merge back into main
      2. commit msgs happens
   2. Rebase flow (rebase in private - FF merge in public)
      1. rebase main into your branch first
      2. then fast-forward merge into main


Merge pull request
Create a merge commit
All commits from this branch will
the base branch via a merge com
Squash and merge
The 1 commit from this branch w
to the base branch.
Rebase and merge
The 1 commit from this branch w
and added to the base branch.
Markdown is supported