EPFL

EMBEDDED SYSTEMS LABORATORY

Introduction to git part 2

EPFL - Embedded Systems Laboratory

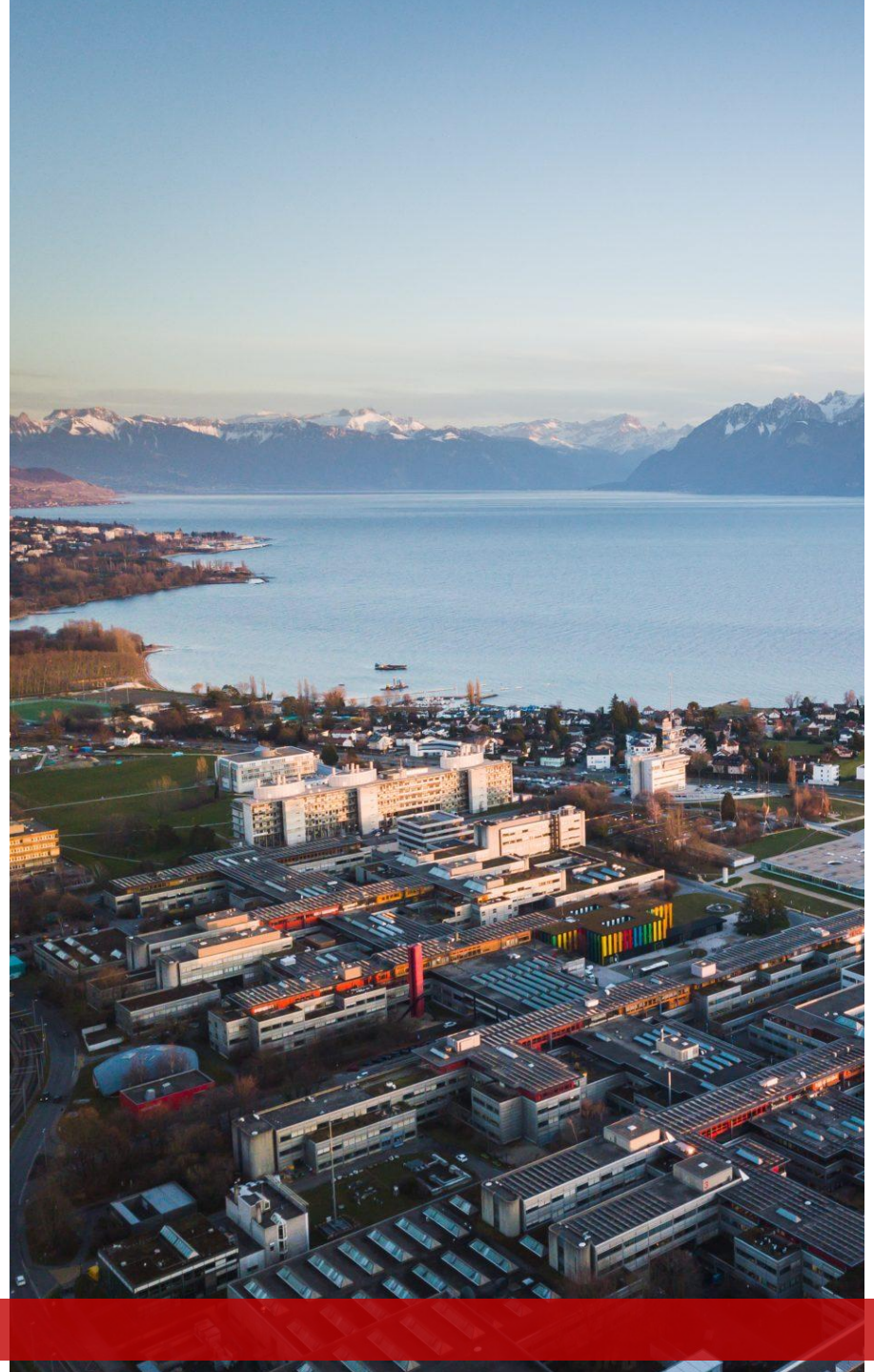
2025

Introduction to git
introduction and basic use

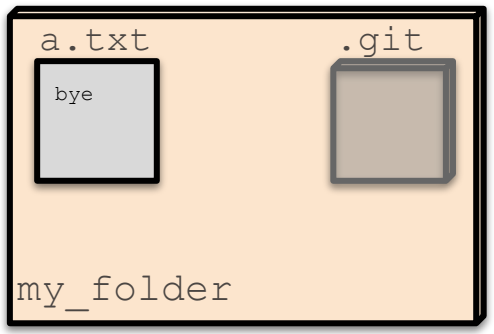
Working alone
for any project you have

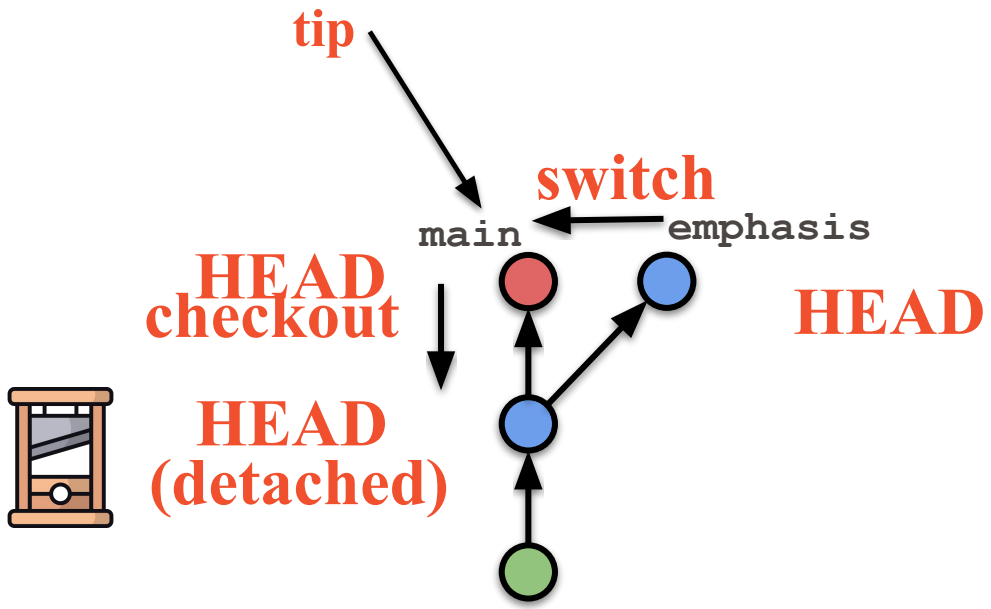
Using other’s work
e.g. the practical works

Working with others
e.g. your final project



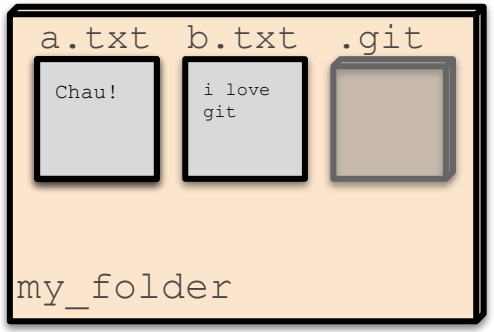
Branches

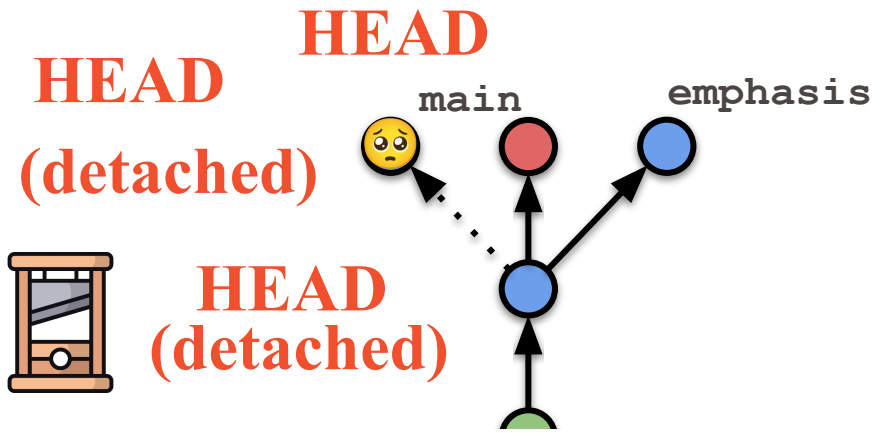

Repository (repo)
a.txt
.git
bye
my_folder


tip
switch
main
emphasis
HEAD
checkout
HEAD
HEAD
(detached)

Branches

Repository (repo)





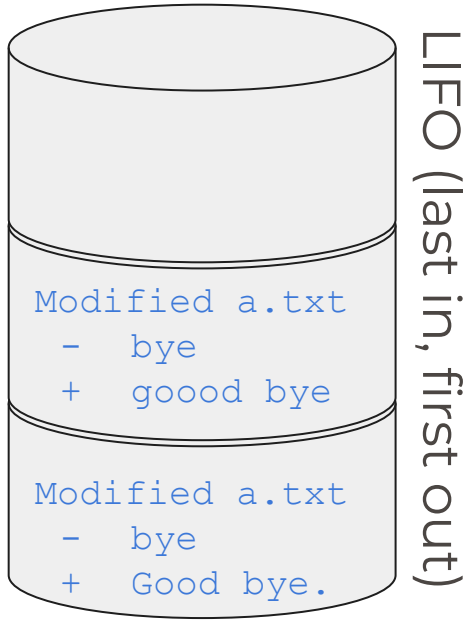
Detached head = 💩: Don't touch, or create a branch!

Changes

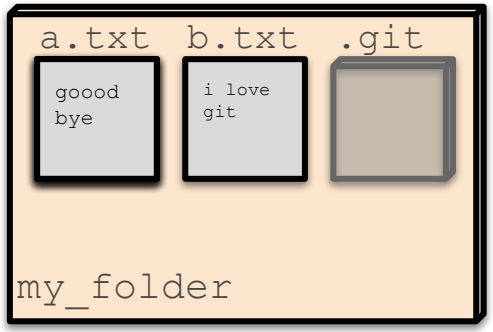
Restore

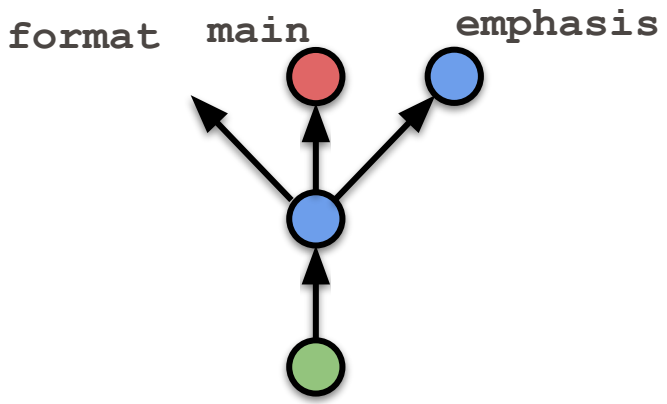
```
Modified a.txt
-  bye
+  goood bye
```

Stash

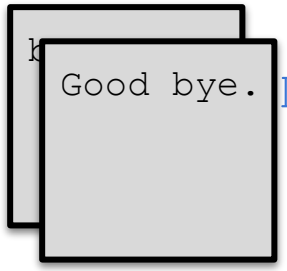


Repository (repo)

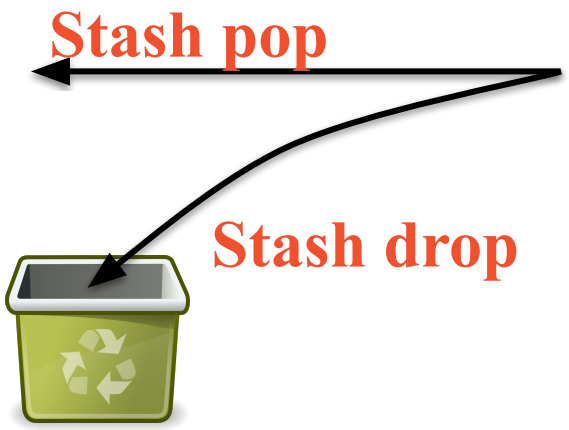


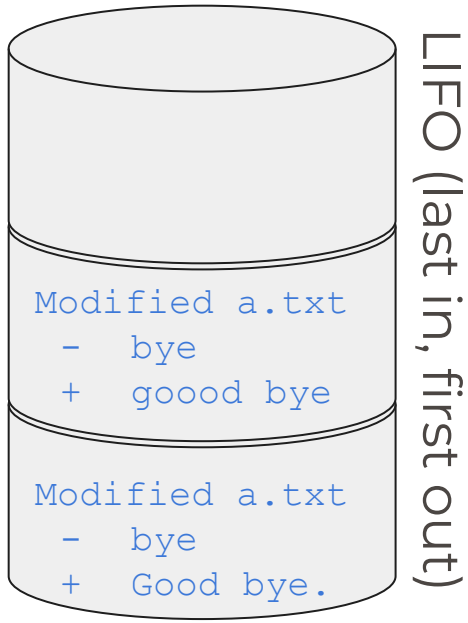


Changes

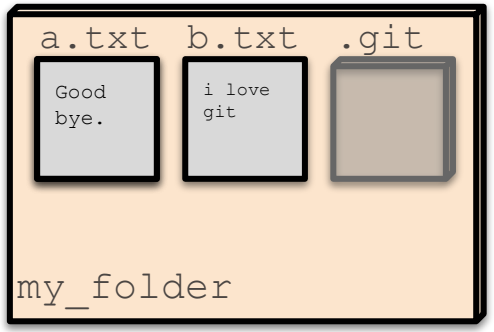


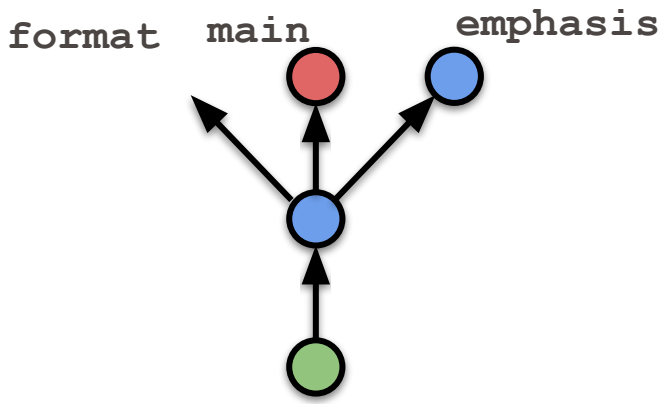

```
Modified a.txt
-  bye
+  Good bye.
```





Repository (repo)

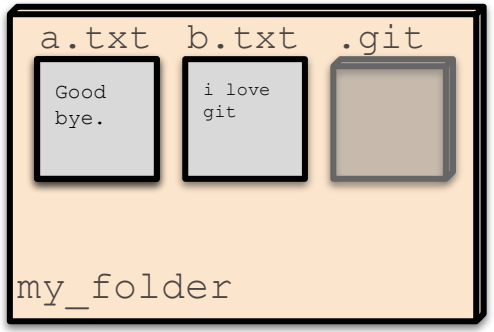


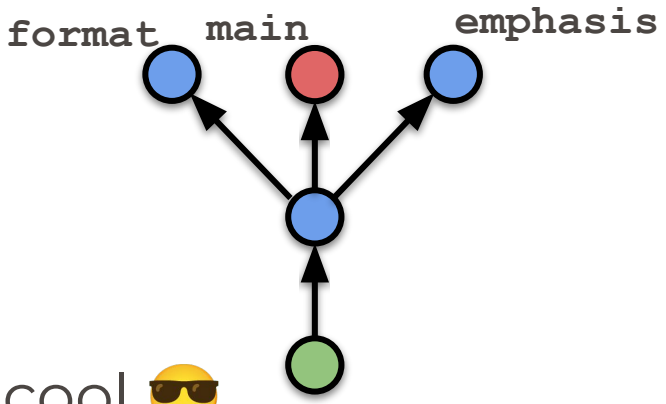


Changes


```
Modified a.txt
- bye
+ Good bye.
```

Repository (repo)



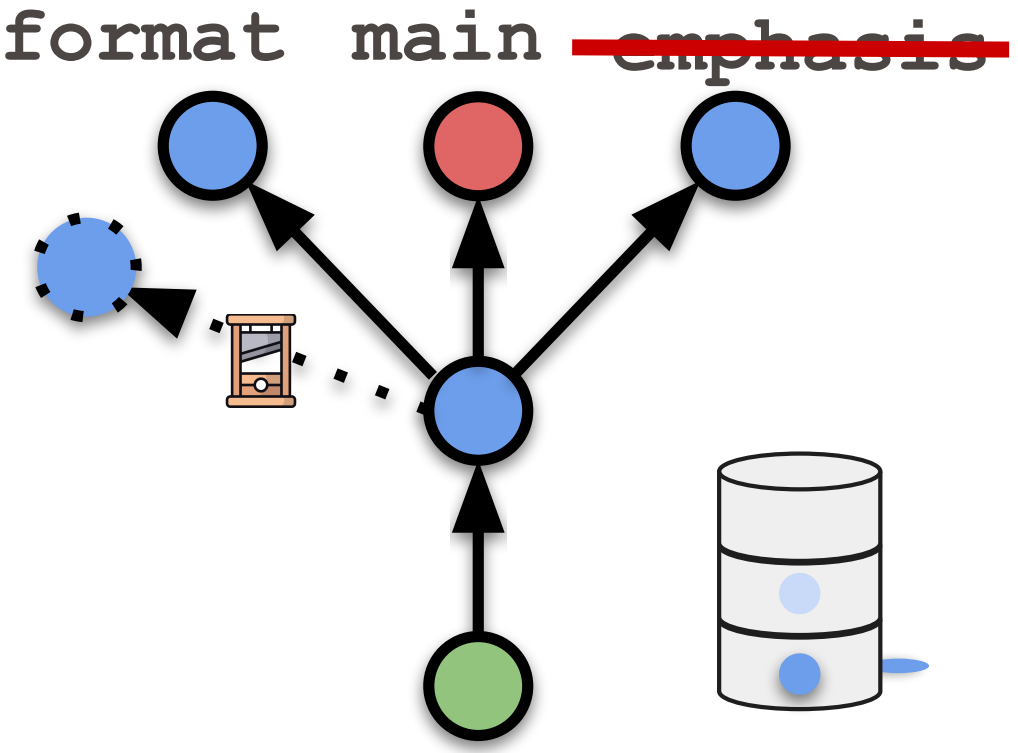


Stash is cool 😎

Demo

Branching

- Go into **detached head** and:
  - Switch back
  - Lose changes
  - Save changes
- Create a proper new branch
- Delete branches
- Recover branches



Changes

- Fail to switch branches
- Stash and pop

Demo - recap

- `git switch <branch name>`
  - Switches your HEAD into a different branch tip
- `git checkout <branch name/commit>`
  - Switches your HEAD into a different commit (can cause detached head!)
- `git branch -D <branch name>`
  - Delete a local branch
- `git checkout -b <new branch name> <branch name/commit>`
  - Creates a new branch called <new_branch_name> from the specified branch or commit

Demo - recap

- `git restore`
  - Removes uncommitted changes
- `git stash`
  - Moves uncommitted changes into the top of the stack
- `git stash list`
  - List the elements in the stack
- `git stash drop`
  - Removes the top of the stack
- `git stash pop`
  - Brings back the top of the stack into uncommitted changes (and removes them from the stack)

Demo - recap

- `ctrl + r <start typing>` then `ctrl + r` or ➡
  - See previous commands and select one
- `ctrl + C`
  - Cancel the command you are writing
- ⬆️
  - Navigate previous commands
- `Select and center click`
  - Paste without copying
- `Ctrl + shift + C/V`
  - Copy/paste in the terminal
- `clear`
  - clear the console

Demo - cheatsheet

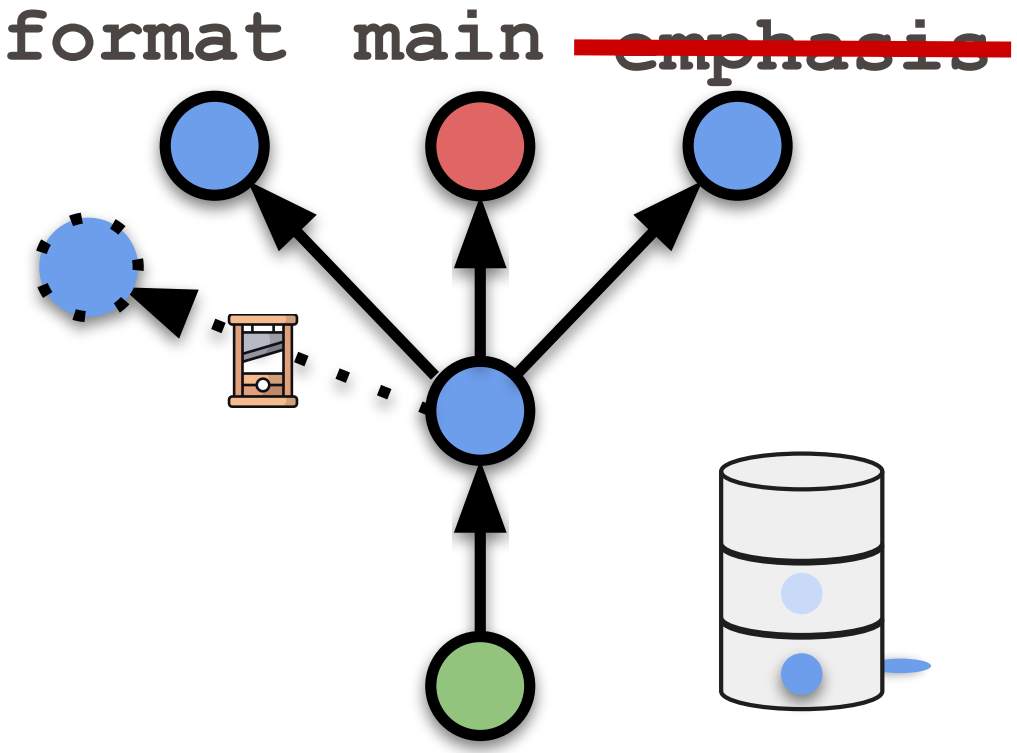
```
        # Go to a detached head and back
git log                                 # To find an older commit
git checkout <hash>                     # Checkout into a detached head
git status                              # Check the status (not the detached head)
git branch -a                           # See all branches, see detached head
git switch main                         # Exit the detached head
git branch -a                           # no trace of our previous detached head

        # Go to a detached head and keep changes
git checkout <hash>
# modify a.txt
git checkout main                       # You can also use checkout instead of switch

        # Go to a detached head and “lose” changes
git checkout <hash>
git add .
git commit -m “Changes to be lost”
git log                                 # See that changes were added to a history, but there is no associated branch!
git checkout main                       # Read the help from git!
git log --all                           # No trace of our commit!

        # Save those changes
git branch saved_branch <suggested hash> # Save the commit into a new branch
git checkout -b also_saved_branch <suggested hash> # Same, but with checkout (and you also checkout, duh…)
git log --all --oneline --graph         # See all the cool branches

        # Delete and recover branches
git log --all                           # See the latest commit from emphasis
git branch -D emphasis                  # Delete the emphasis branch
git checkout -b emphasis2 <hash from emphasis>  # Recover the branch
```

```
        # Clean the repo before going ahead
git branch -a                           # List all branches
git checkout main                       # Switch to main
git branch -D some_branch some_other_branch  # Delete the two useless branches
git branch -m emphasis2 emphasis        # Rename the saved branch
git branch                              # Check everything is how i want it

        # Do some changes into a new branch
git log                                 # To check which hash I want to checkout to
git checkout -b format <hash>           # Create a new branch called “format”
# Make some changes in b.txt, which does not exist in main!
# git checkout main                     # See how git will not be able to switch to main

        # Try to overcome this limitation with stash
git stash                               # Move my changes to the stack
git stash list                          # List all my stashes
# Do some more modifications
git stash                               # Stash the new changes
git stash list                          # See how the indexes were updated
git stash drop                          # Discard the latest change in the list
git stash list                          # See how the latest change disappeared
git checkout main                       # Now the switch is effective, but there is a CONFLICT
# Note that the conflict was self-solved by git, which gave prevalence to the stashed change.
git status                              # See the changes that were applied
# Note that “us” is the content of the branch main, not the changes staged
git add .                               # Move the changes to the staging area (index)
git status                              # See how b.txt is considered a new file
# We could commit this change, but that would be easy… let’s discard it!
git restore --staged b.txt              # Remove the added file from the staging area
# Now b.txt is untracked! So git cannot do any operation over it other than adding it.
rm b.txt                                # Remove b from the working directory
```

Demo - try it yourself

Branching

- Go into **detached head** and:
  - Switch back
  - Lose changes
  - Save changes
- Create a proper new branch
- Delete branches
- Recover branches



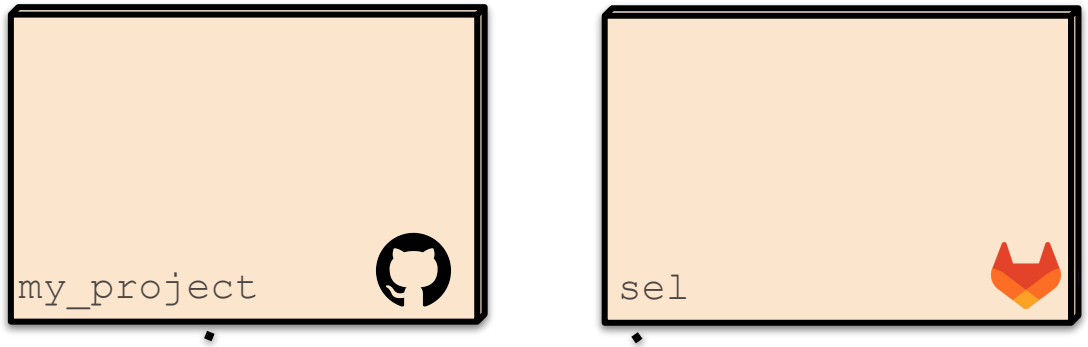
Changes

- Fail to switch branches
- Stash and pop

Remote repos

Bitbucket
GitLab
GitHub

Remote repo



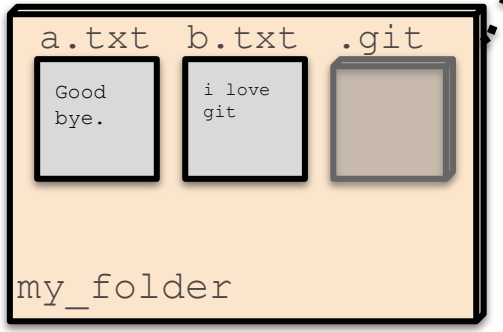
Remote

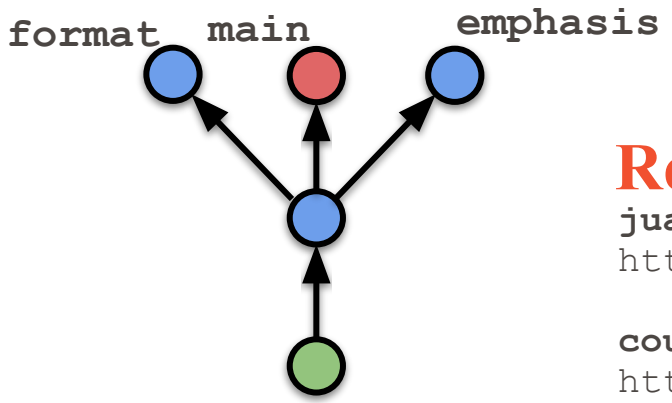
Local

juan

course

Repository (repo)





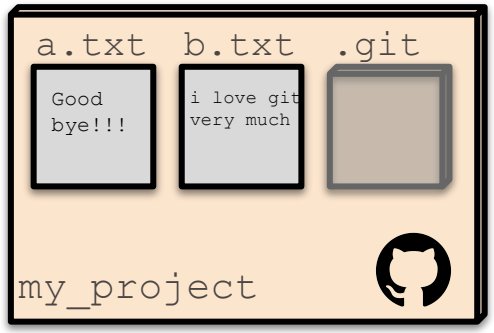
Remotes' list

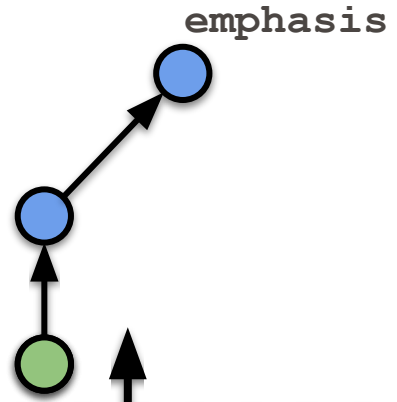
juan
https://github.com/juan/my_project.git

course
https://gitlab.epfl.ch/sel/pws.git

Remote repos

Remote repo



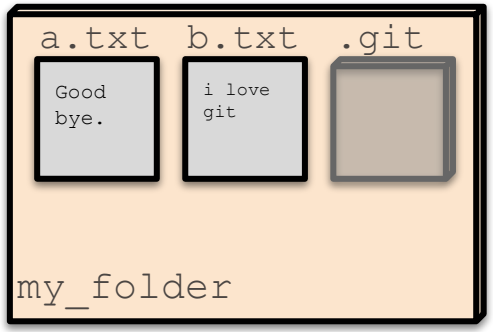


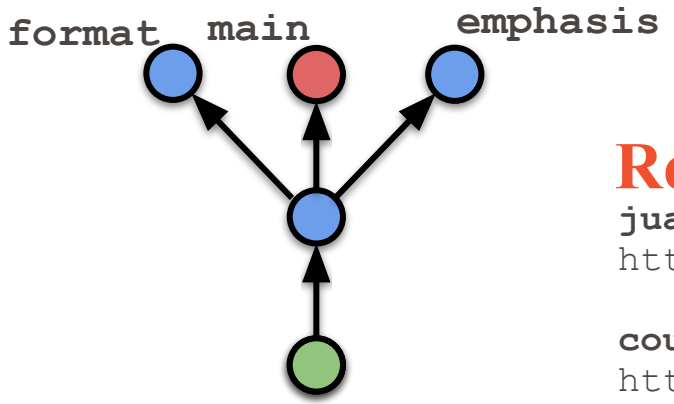
Remote

Local

Push

Repository (repo)





Remotes' list

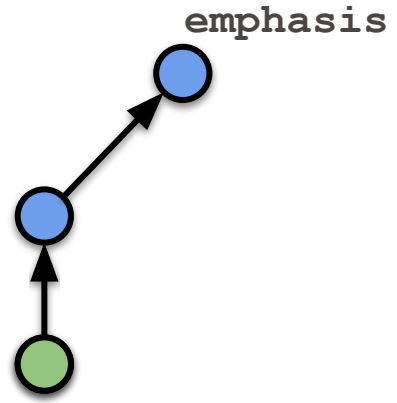
juan
https://github.com/juan/my_project.git

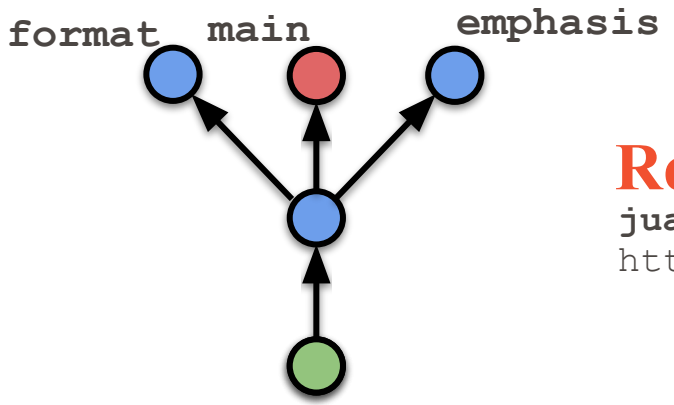
course
https://gitlab.epfl.ch/sel/pws.git

Demo

Remote repo

- Create an empty remote repo
  - Giving you the right accesses
  - Setting up an SSH key
- Push a repo



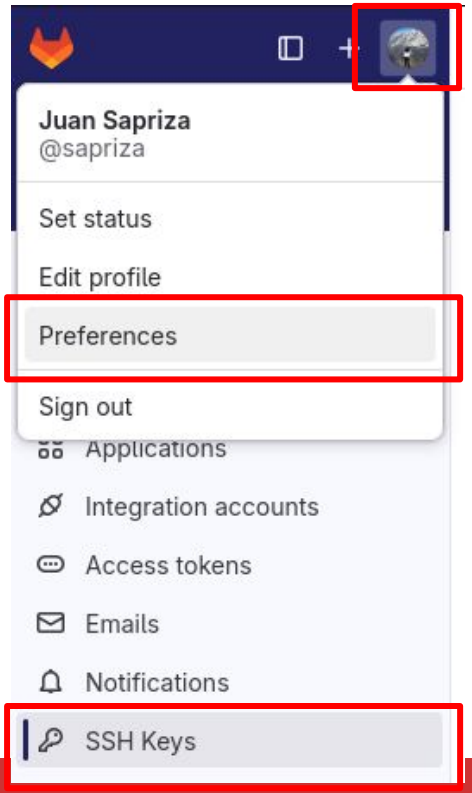


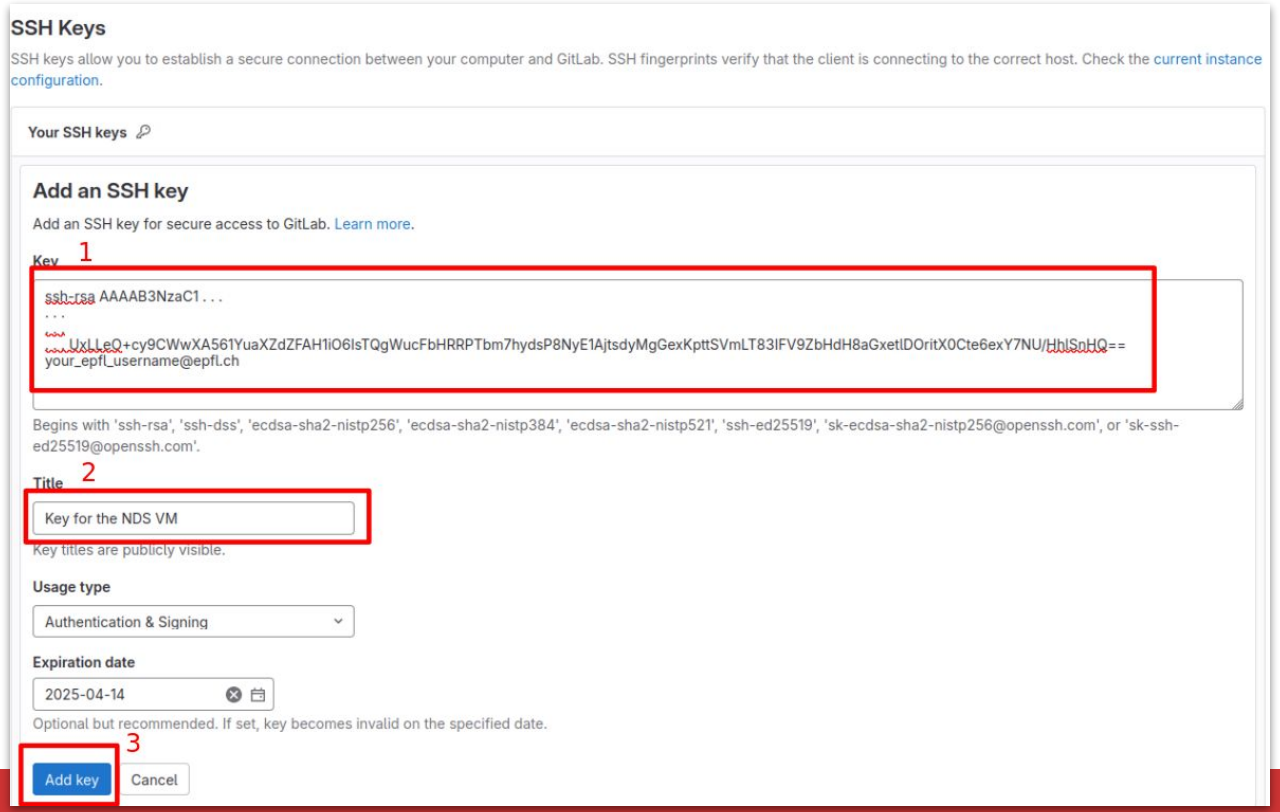
Remotes' list

juan
https://gitlab.epfl.ch/my_proj.git

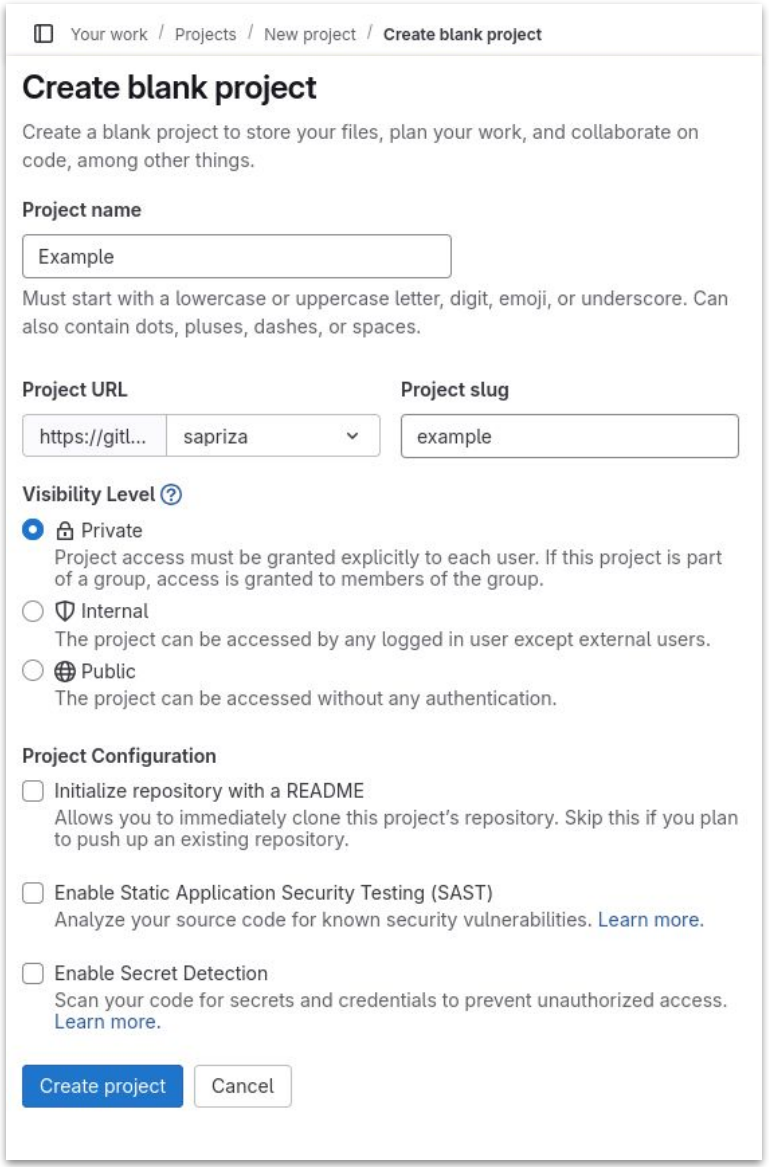
Demo - recap

- `ssh-keygen -t rsa -b 4096 -C "your epfl username@epfl.ch"`
  - Generates an SSH key for this email in this computer.
- `cat ~/.ssh/id_rsa.pub`
  - Prints the generated SSH key

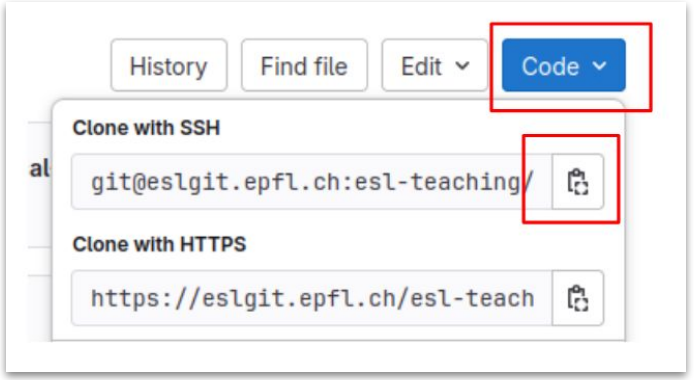




Demo - recap



- `git remote add <remote name> <url>`
  - Adds a remote to the remote list
- `git branch -a`
  - Shows **a**ll branches (local and remote)
- `git push <remote name> <branch name>`
  - Update the remote branch with the changes from the local branch



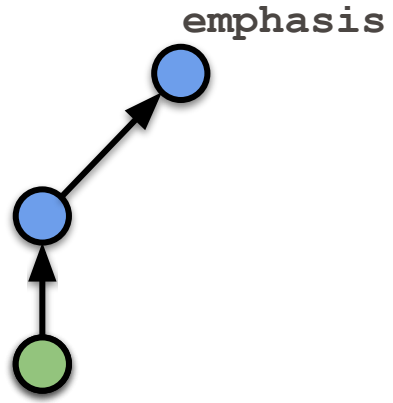
Demo - cheatsheet

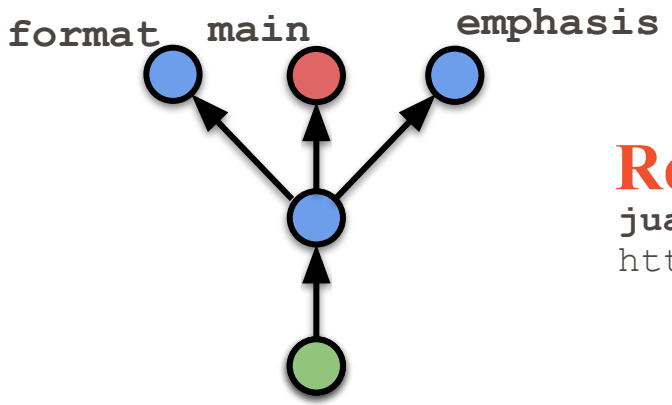
```
        # Authenticate yourself on gitlab
ssh-keygen -t rsa -b 4096 -C "your_epfl_username@epfl.ch"
# enter, enter, enter
cat ~/.ssh/id_rsa.pub
# Go to gitlab
# Click on your picture > preferences > SSH keys > add new key
# Copy and paste the long character sequence (your public key) and paste it in the field
# Change the name to identify where you run these commands (e.g. your laptop, the labs machine, the VM, …)
# Save

        # Create a new repo
# On gitlab, click on the + sign next to your picture > new project/repository
# Select “blank project”, give it a name, and make sure you uncheck “Initialize with a README”
# Go to the blue button that says “code” and copy the first (SSH) url

        # Add a remote
git remote add personal <url>       # Create a new link to a remote called “personal”
git push personal emphasis          # Push the branch emphasis to the remote “personal”
git branch -a                       # Check that there are remote branches as well now

        # Add some commits to get out of sync
# Make some changes
git commit -am “Some changes that will not be pushed yet”
git log                             # Note the difference between local and remote branches
```

Demo - try it yourself

Remote repo

- Create an empty remote repo
  - Giving you the right accesses
  - Setting up an SSH key
- Push a repo



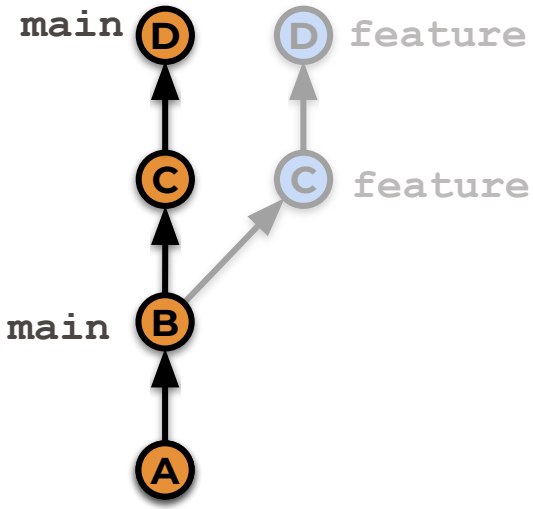


Remotes' list

juan
https://gitlab.epfl.ch/my_proj.git

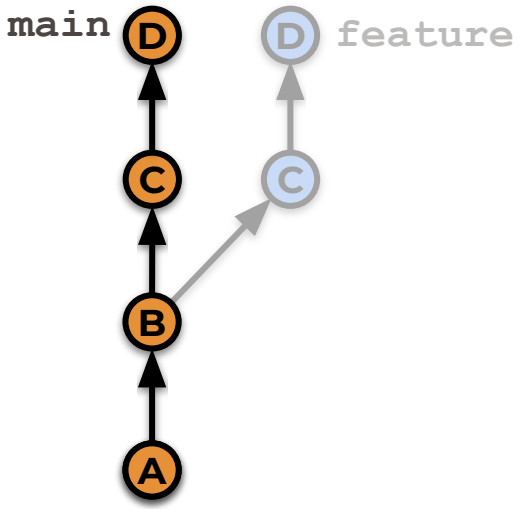
Joining branches

Merge (fast-forward)

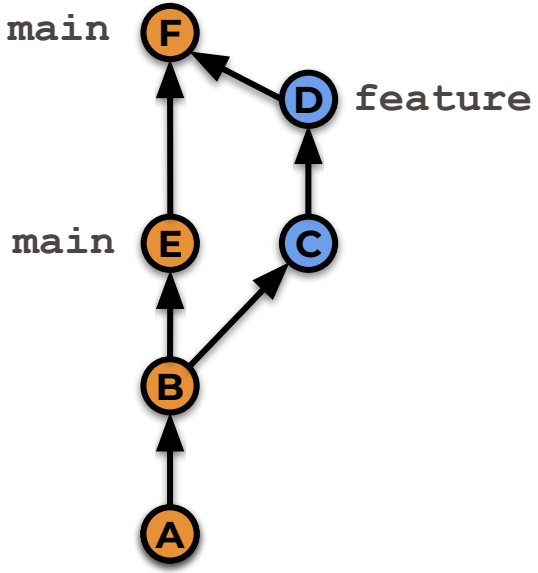


Joining branches

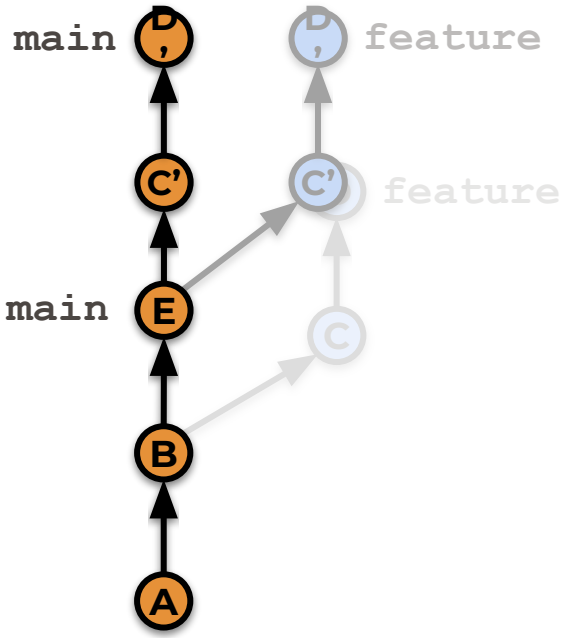
Merge (fast-forward)



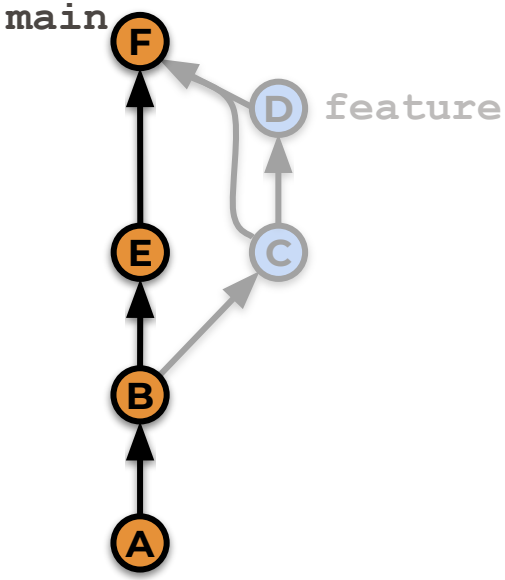
Merge



Rebase +fast-forward



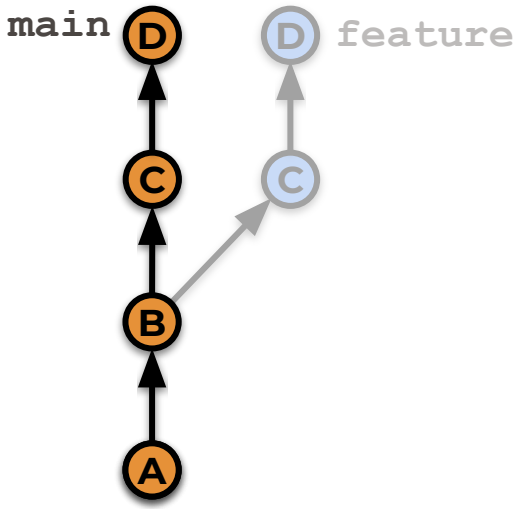
Squash commit



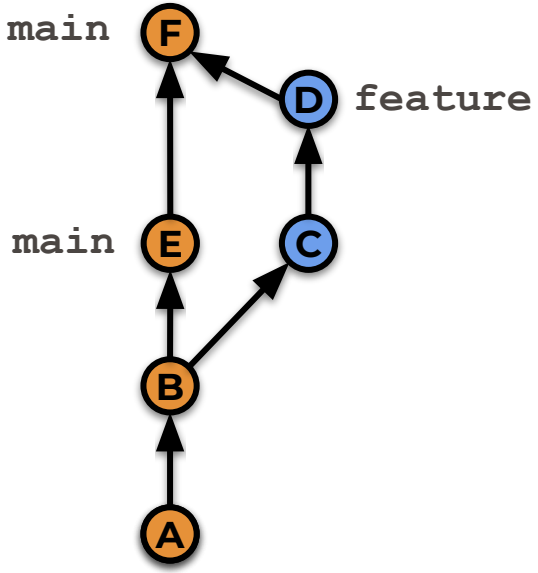
If in doubt: **merge**

Demo

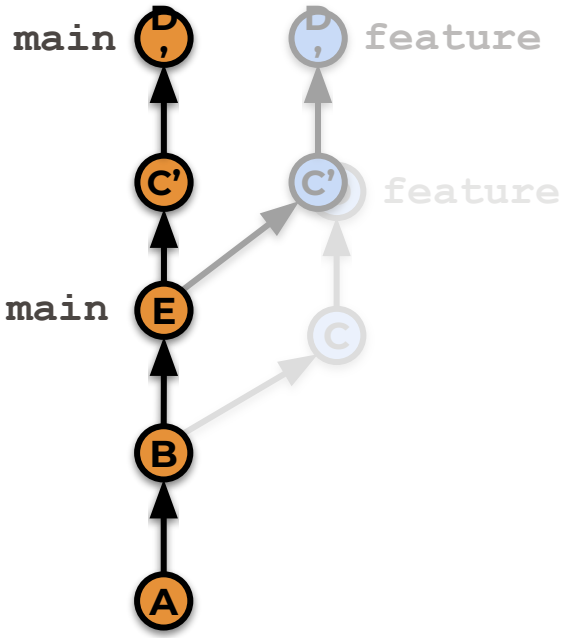
Merge (fast-forward)



Merge



Rebase +fast-forward



Demo - recap

- `git merge <branch>`
  - Merge the commits of <branch> into HEAD
- `git rebase <branch>`
  - Removes all commits from our HEAD until the last common commit with <branch>
  - Applies the missing commits from <branch>
  - Recreates the commits we had in HEAD
  - **Does not merge into <branch>!**
- `git merge/rebase --abort`
  - “Aaaaaaaaaaaahhh go back go back go back”
- `ctrl + x`
  - Exit the nano editor

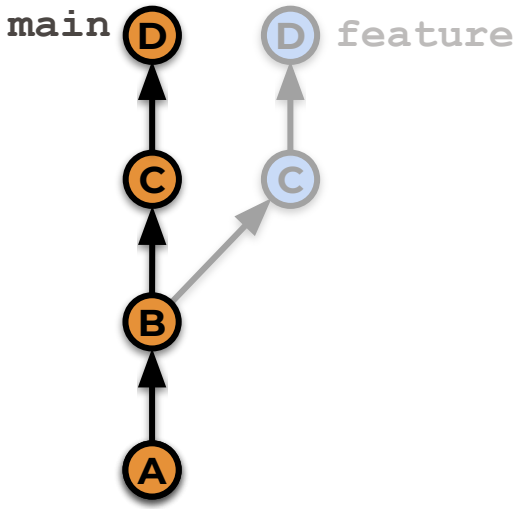
Demo - cheatsheet

```
        # Try to merge emphasis
git log --all --oneline  --graph                # Note that we have a feature branch “emphasis”
git checkout main                               # In case we were not on main already
git merge emphasis                              # Try to merge emphasis into main
# Note that there is a CONFLICT: there is no b.txt on main, but the commit on emphasis tries to apply a change to it
# Let’s not try to deal with this YET.
git merge --abort                               # Cancel the merge attempt

        # Create a new feature branch and merge it into main
git checkout -b feature
# Make changes to a.txt
git add .
git commit -m “Developed a new feature in a.txt”
git checkout main                               # Go back to the main branch
git merge feature                               # Merging should be fast-forwarded
git log                                         # See the commit merged into the history of main

        # Create a new branch and merge with changes on both sides
git checkout -b other_feautre
# Make changes to a.txt
git commit -am “Developed another feature”      # For when you are tired of running git add . before
git checkout main
# Make some changes that do not conflict with other_feature, for example, adding c.txt
git add c.txt
git commit -m “Added a completely different thing”
git merge other_feature                         # Try to merge the feature branch
# This will open nano, a text editor.
# You can modify the commit message there
# When you are happy, press ctrl+x

git log --oneline  --graph                      # See that the history of THIS branch is not linear anymore

        # Make VSCode your default editor
git config --global core.editor "code --wait"
git checkout -b test
# make a change
git commit -am “test 1”
git checkout main
# make a change
git commit -am “test 2”
git merge test
# Edit the commit message on vscode and close the window
git log  --oneline --graph
git branch -D feature other_feature test        # Clean the repo
```

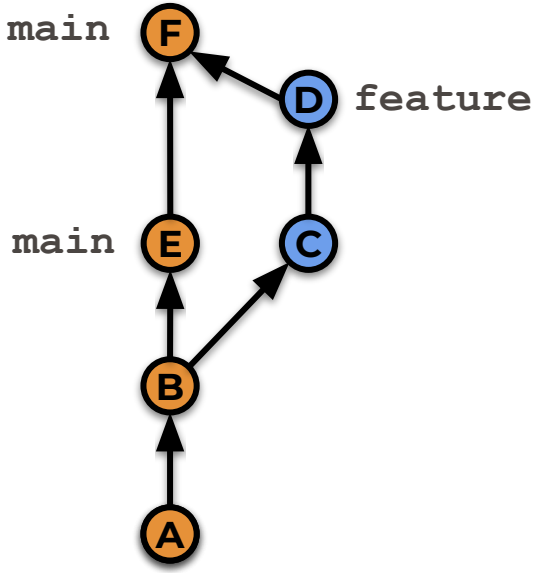
```
git checkout -b feature
# Add a new file f.txt with a feature
git commit -am “Added a new feature in f”
git log                                         # Take note of the hash of this commit
git checkout main
# Fix a bug in a.txt
git commit -am “Fix a bug in a”
git checkout feature
git rebase main                                 # Everything should go fine
git log                                         # Note that the hash is different!

git checkout main
git merge feature
git log --online --graph                        # Note how now the history is linear (unlike with the merge)
```

Demo - try it yourself

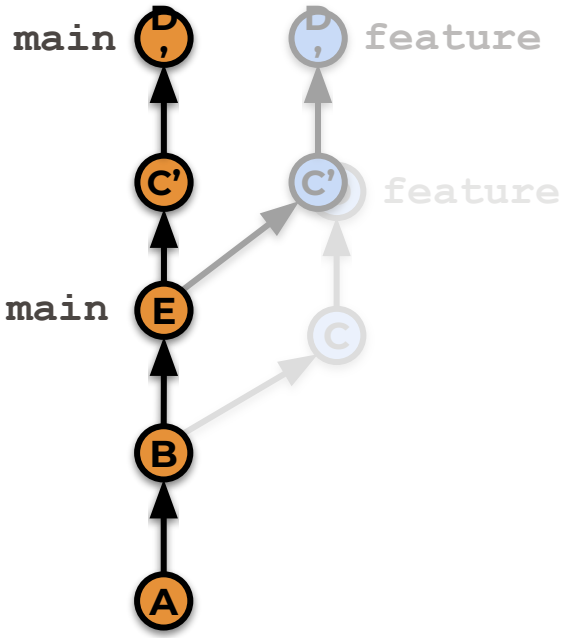
Merge (fast-forward)



Merge



Rebase +fast-forward



Thank you!

EPFL - Embedded Systems Laboratory