Introduction to Open & Reproducible Science (IORDS)

Michael Dayan, Data Scientist Manager

Methods & Data facility
Human Neuroscience Platform
Foundation Campus Biotech Geneva

Virtual machine info

To get an IP, please fill the form at:
`https://tinyurl.com/IORDS2021-IP-git3`

START RDP CLIENT (as instructed in email / Slack):

- *Remote Desktop Connection* on Windows
- *Remote Desktop App* on Mac OS
- *Remmina* on Linux distributions (e.g. Ubuntu)

Connecting your:	WIFI SSID	WIFI Password
Laptop (no phones)	NIDS_course	reproduciblescience
Phone	CAMPUS_VISITORS	welcomecampus

PLEASE CONNECT TO THE VM
→ Login: brainhacker
→ Password: brainhack!

Connect to Slack and download the exercise slides

ANY PROBLEM? Please raise your hand or ask questions on Slack: channel **`#git`**

On site support (including coding):


Nathan


Louis

Remote support (including coding):


Serafeim

HNP METHODS & DATA

LECTURE OBJECTIVES

GIT lectures objectives (you should be able to…):

- Know what GIT is useful for and identify some famous actors in the Git ecosystem
- Understand the “promotion model” and “commits” at the heart of GIT
- Describe the typical workflow to record code changes
- Examine the code history and compare different code versions
- Understand the concepts of branches and branch merging

GIT Part 1

- Know how to collaborate with Git using Github
- Understand the concepts of remote repository / remote branches
- Synchronize code changes between local and remote repositories
- Distinguish between different kinds of merge

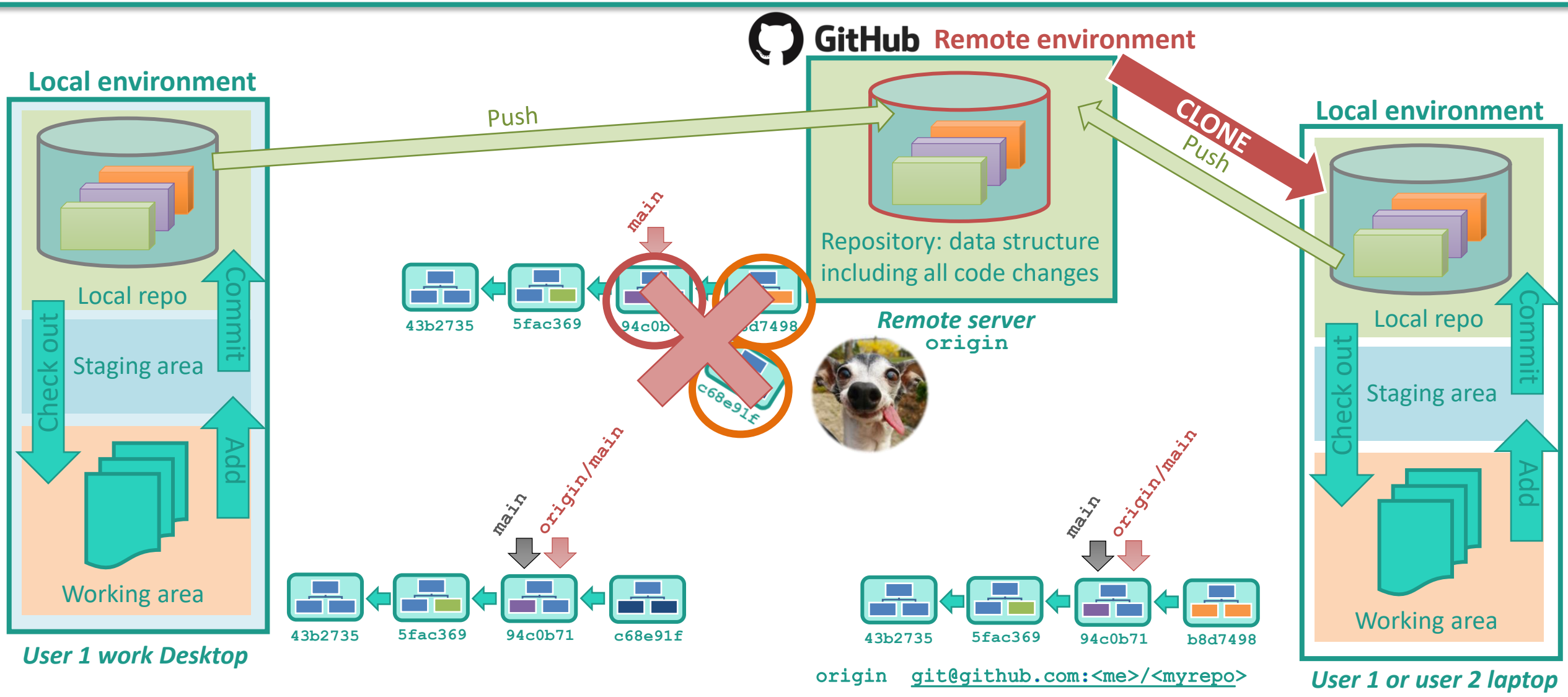
GIT Part 2

- Know advanced merging techniques and deal with conflicts
- Collaborate on public projects with Github

GIT Part 3

→ Track your code changes with GIT and collaborate on your own or public projects with Github

GIT AND GITHUB RECAP



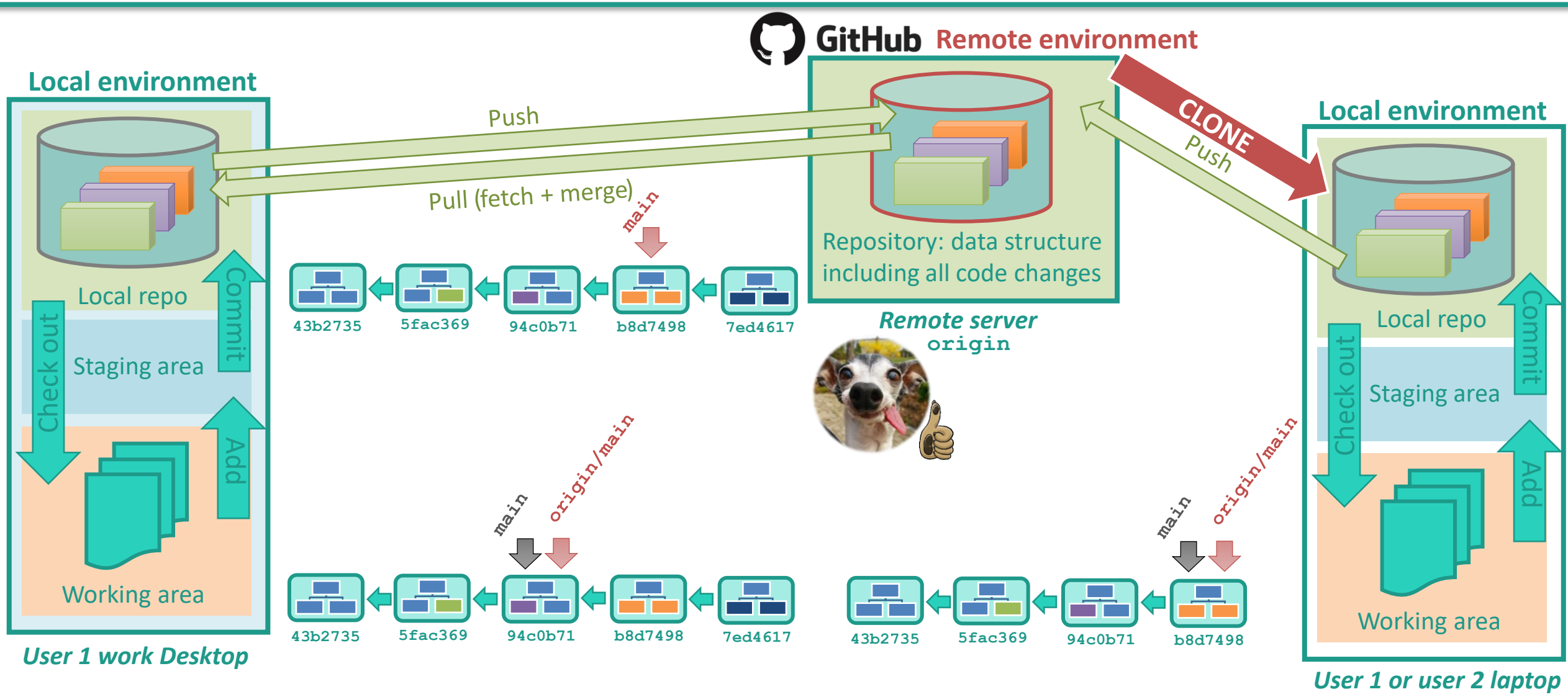

```
git remote add origin git@github.com:<me>/<myrepo>.git
git push -u origin main
```

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

GIT AND GITHUB RECAP



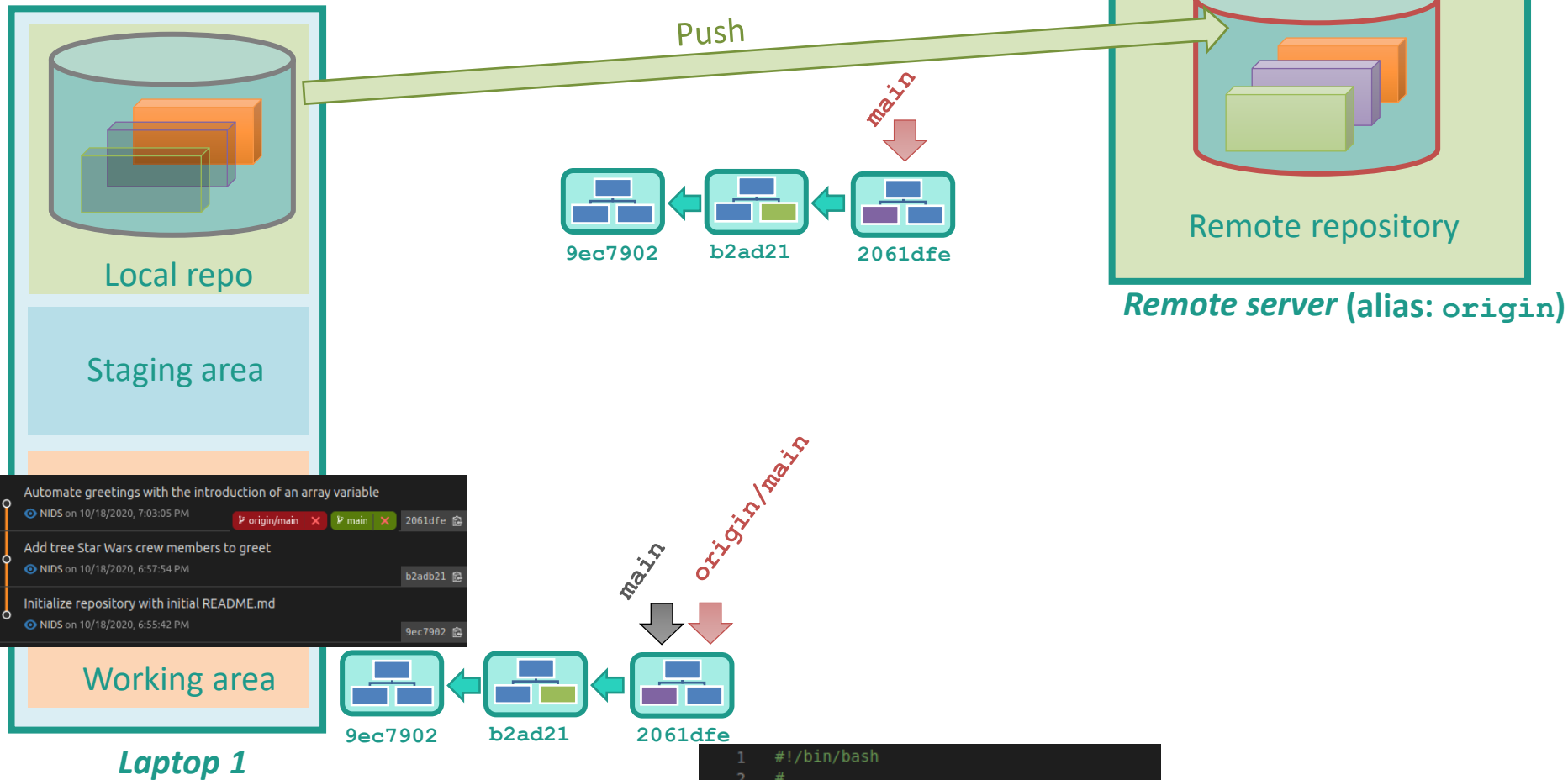

```
git fetch origin main
git merge origin/main
```
same as `git pull origin main`

```
git add .
git commit -m "add feature"
```
`git push origin main`

GIT AND GITHUB RECAP

Before starting lab: local repo (in `laptop1`) with 3 commits

Local environment



Laptop 1

GitHub Remote environment

Remote server **(alias: `origin`)**

After part 1:

- an empty GitHub repo

After part 2:

- a GitHub repo with the latest version on laptop 1
- a local tracker of the remote branch `main`

```bash
#!/bin/bash
#
# Automated greeter for Star Wars crew

SW_CREW=("Leila" "R3D3" "Han Duo")

for crew_member in "${SW_CREW[@]}"; do
  echo "Hi ${crew_member}"
done
```

GIT AND GITHUB RECAP

Before starting lab: local repo (in `laptop1`) with 3 commits

GitHub **Remote environment**

❖ **on GitHub : an update**

Local environment

Local repo

Push

main

9ec7902 b2ad21 2061dfe 5f18820

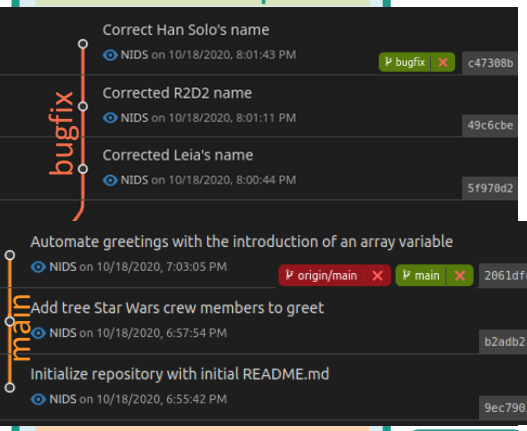
Remote repository

CLONE

Push

Local environment

Remote server (alias: `origin`)



After part 3:

❖ **on laptop 1: a new branch `bugfix`**

❖ **on laptop 2:**

- **a clone WITHOUT `bugfix`**
- **a new commit on `main`**

bugfix

main origin/main

bbe7484 82e75af 59902c5

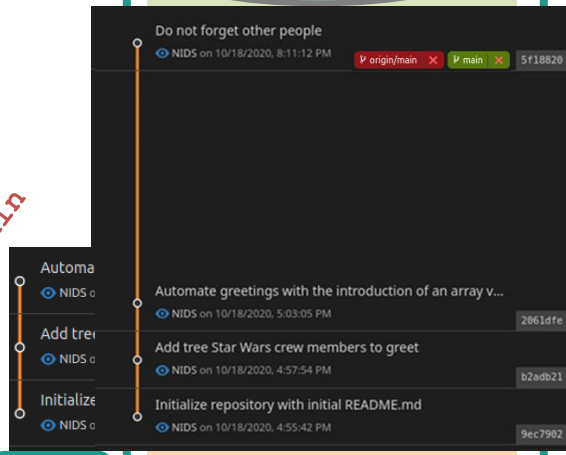
9ec7902 b2ad21 2061dfe

Working area

Laptop 1

main origin/main

9ec7902 b2ad21 2061dfe 5f18820



`R.md hello`

Working area

Laptop 2

```bash
#!/bin/bash
#
# Automated greeter for Star Wars crew

SW_CREW=("Leia" "R2D2" "Han Solo")

for crew_member in "${SW_CREW[@]}"; do
  echo "Hi ${crew_member}"
done
```

```bash
#!/bin/bash
#
# Automated greeter for Star Wars crew

SW_CREW=("Leila" "R3D3" "Han Duo")

for crew_member in "${SW_CREW[@]}"; do
  echo "Hi ${crew_member}"
done

echo "Hi everyone"
```

HNP METHODS & DATA

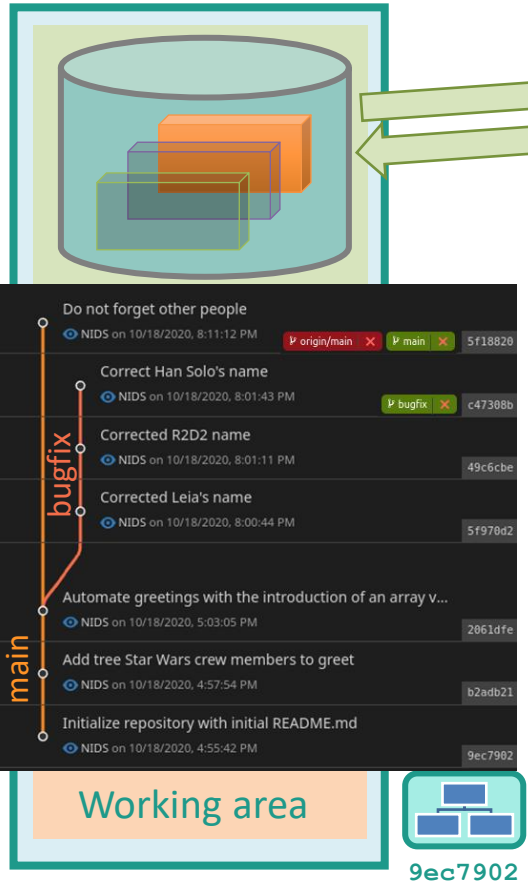
campus biotech

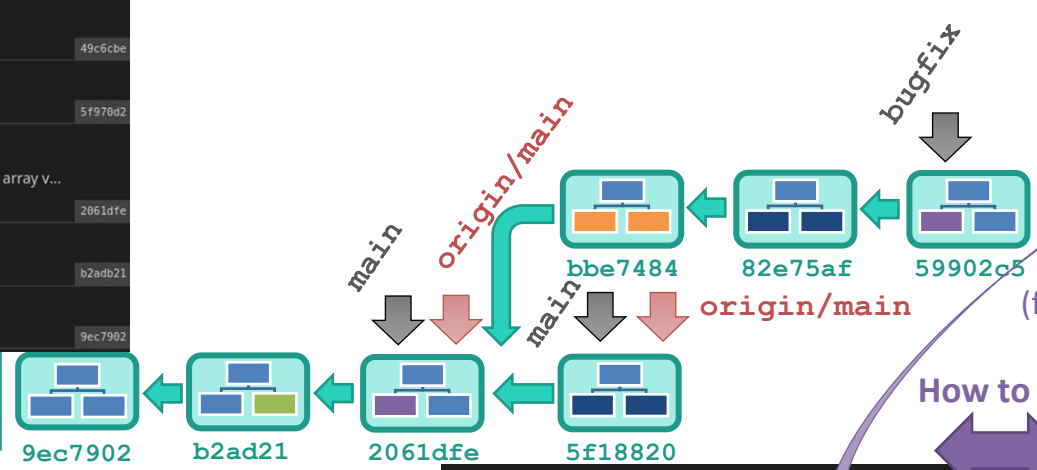
Fondation Campus Biotech Geneva

GIT AND GITHUB RECAP

Before starting lab: local repo (in `laptop1`) with 3 commits

Local environment



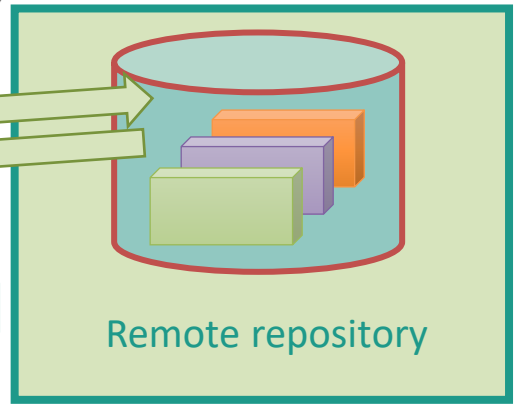


Laptop 1

Push

Pull (fetch + merge)

GitHub Remote environment



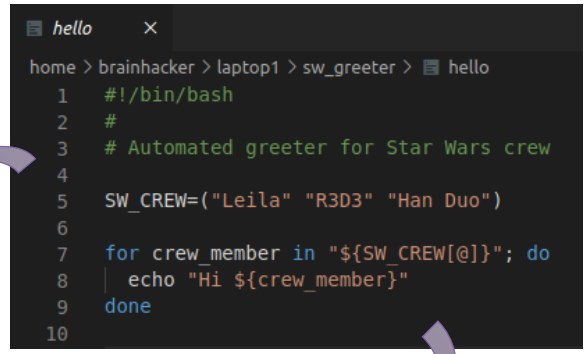
Remote server (alias: `origin`)

After part 4:

- on laptop 1:
  - an update of the `main` remote branch tracker (`origin/main`)
  - a merge of remote `main` (`origin/main`) with local `main`

Note: could have done both simultaneously with `pull`

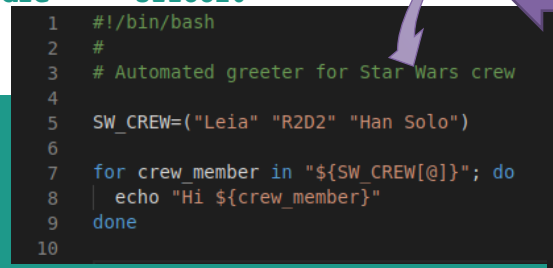
Through `bugfix` (from laptop 1)

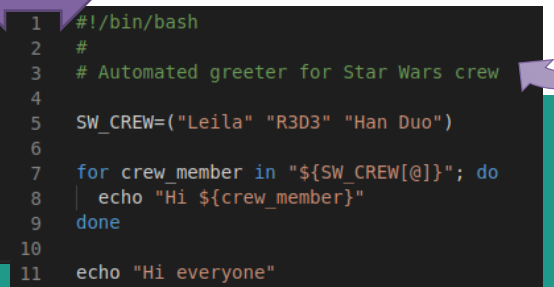


Through Github latest `main`

(GitHub was last updated from laptop 2 which had no idea of `bugfix`)

How to merge ?



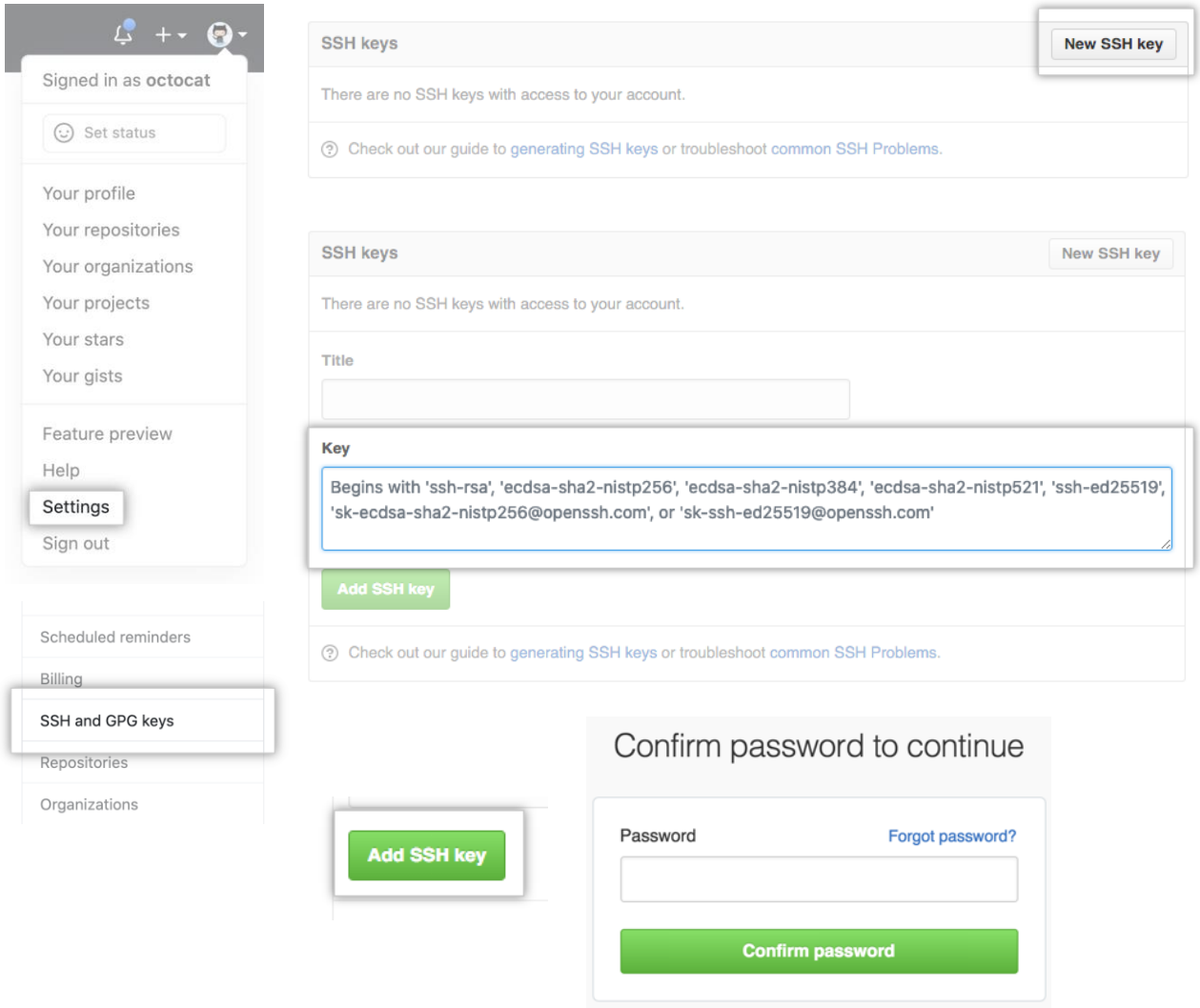


HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

How to add your public SSH key to your GitHub account ?



Authorizing your computer to access GitHub

This is done by copying your local user public SSH key to the "authorized keys" on GitHub

- On your local computer, display the content of your public key on the terminal and copy it (you may need create a public/private key pair if you don't have one already)

```
cat ~/.ssh/id_rsa.pub
```

  To copy it you could select the output with your mouse, right click and select copy (or use the SHIFT + CTRL + C shortcut)

- On GitHub:
  - Click on your profile photo then on Settings
  - Click on "SSH and GPG keys" in the sidebar
  - Click on "New SSH key" or "Add SSH key", and
    - Add a label for your computer (e.g. "Home Laptop")
    - Paste the content of your public key in the "Key" field
  - Click on "Add SSH key"
  - Confirm your GitHub password if asked

GIT LAB 1 – setup local laptop 2 repo and GitHub repo

You are using a new VM for this lecture, so you need to:

- Indicate to git your name and email (please use email you registered to the course with). Note: this is to be shown in the commit history and to compute the commit identifiers (SHA). It does not have to match anything on GitHub.
- Create an SSH key pair
- Display the SSH public key (`~/id_rsa.pub`) on the terminal
- Add this public key to your GitHub account (cf previous slide for help)

Create a clean GitHub repo on push the content of laptop 2

- Create a new repo on GitHub, call it `starwars_greeter2`, and make sure to note its SSH location
- Navigate to the `~/laptop2/starwars_greeter` directory
- Create the remote (i.e the alias) `origin` to point to the remote repository location you just noted
- Push to the GitHub remote `origin` the branch `main` so that to track the remote branch locally (tip: use the `-u` flag)

```
git config --global user.name NAME  # use quotes ("") for NAME
git config --global user.email EMAIL
pwd # print path of current directory
cd DIR # change to directory DIR (i.e. go inside DIR)
       # note: ~ = ${HOME} = /home/brainhacker
ls # list files and directories in current dir
ssh-keygen -t rsa -b 4096 # create an SSH key pair (keep
                          # pressing the "Enter" key to
                          # select all default options
cat FILE # display the content of the file FILE
```

Note: the git SSH location you need to write down has format `git@github.com:<user>/<repo>.git`

```
git remote add ALIAS git@github.com:<user>/<repo>.git
 # set alias ALIAS to a GitHub remote repository location
git remote -v # list all remotes
git push -u ALIAS BRANCH
# push to remote repo ALIAS the branch BRANCH
# and track the remote BRANCH locally (-u option)
```

Check you can see all the files in your new GitHub repo

GIT LAB 1 – setup local laptop 1 repo

Update the remote server location for laptop 1:

Navigate to the `~/laptop/sw_greeter` directory

Create the remote (i.e the alias) `origin` to point to the same remote repository location you noted previously

Visualize the repo history on `laptop 1` local repo

Navigate to the `~/laptop1/sw_greeter` directory

Activate the branch `main`

At the last lecture we pushed `main` to GitHub so the local `main` was tracking the remote `origin/main` branch. Since we created a new GitHub repo we need to restore the tracking. Do this with: `git branch -u origin/main`

Pull the latest changes from the branch `main`

Activate the branch `main` and check the commits history

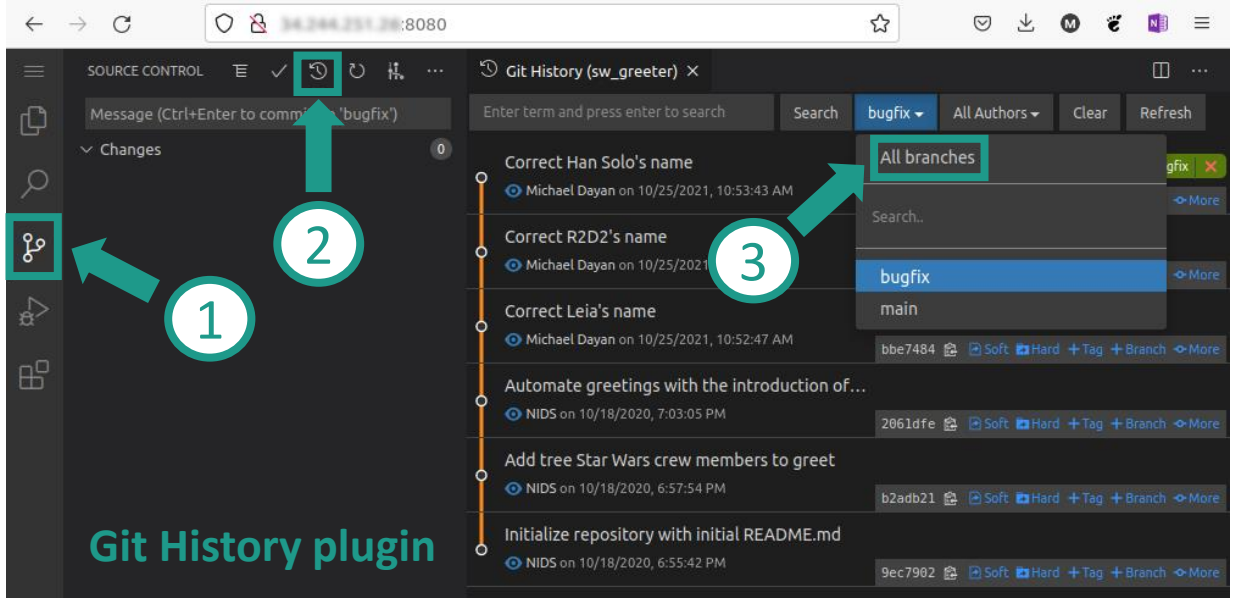
Open a VS Code tab on your browser and open the folder `~/laptop1/sw_greeter`

Visualize the commit graph history in VS Code (cf help in next slide)

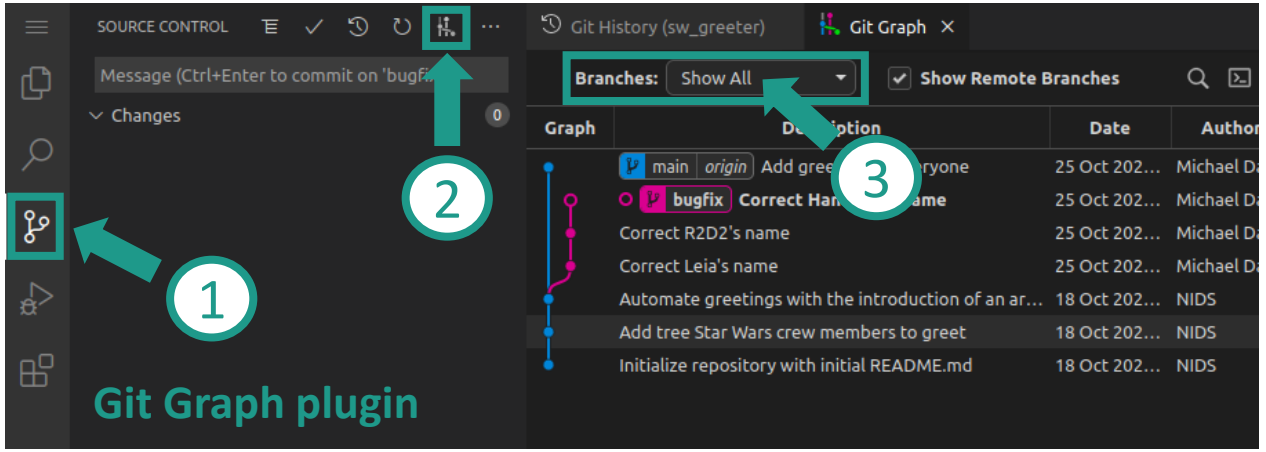
```
pwd # print path of current directory
cd DIR # change to directory DIR (i.e. go inside DIR)
       # note: ~ = ${HOME} = /home/brainhacker
ls # list files and directories in current dir

git remote add ALIAS git@github.com:<user>/<repo>.git
 # set alias ALIAS to a GitHub remote repository location
git remote -v # list all remotes
git checkout BRANCH # activate the branch BRANCH
git branch -u ALIAS/BRANCH # set the current branch to track
                           # at ALIAS the remote branch BRANCH
git pull # pull from the GitHub repo the remote branch
         # tracked by the current active branch
git checkout BRANCH # activate the branch BRANCH
git log --oneline # get concise commit history
To open VS Code in an internet browser, type <YOUR IP>:8080 in
the address bar (password: braincode!).
To open a folder in VS Code: "File" → "Open Folder". Choose
the right folder, then click "OK".
```

GIT LAB 1 – setup & checking commit history

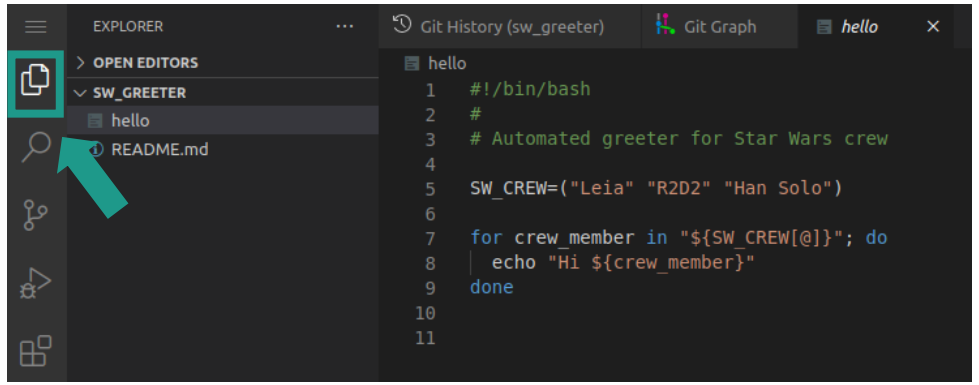
To visualize the commit history in VS Code, two possible plugins:



1. Click on the “source control icon”
2. Click on the Git History plugin
3. (Optional) Select “All branches”



1. Click on the “source control icon”
2. Click on the Git History plugin
3. (Optional) Select “Show All”

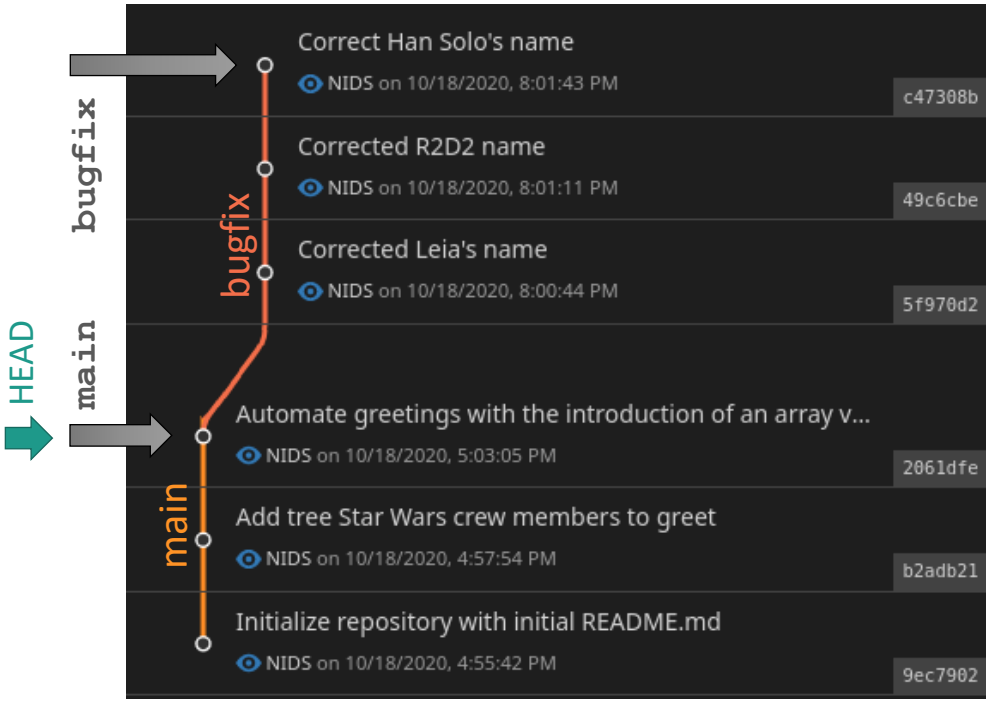


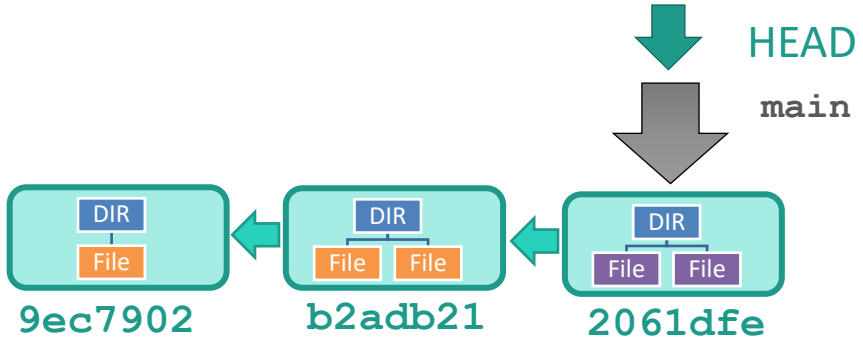
To go back to your files,
click on the “Explorer” icon

MERGING – FAST-FORWARDING

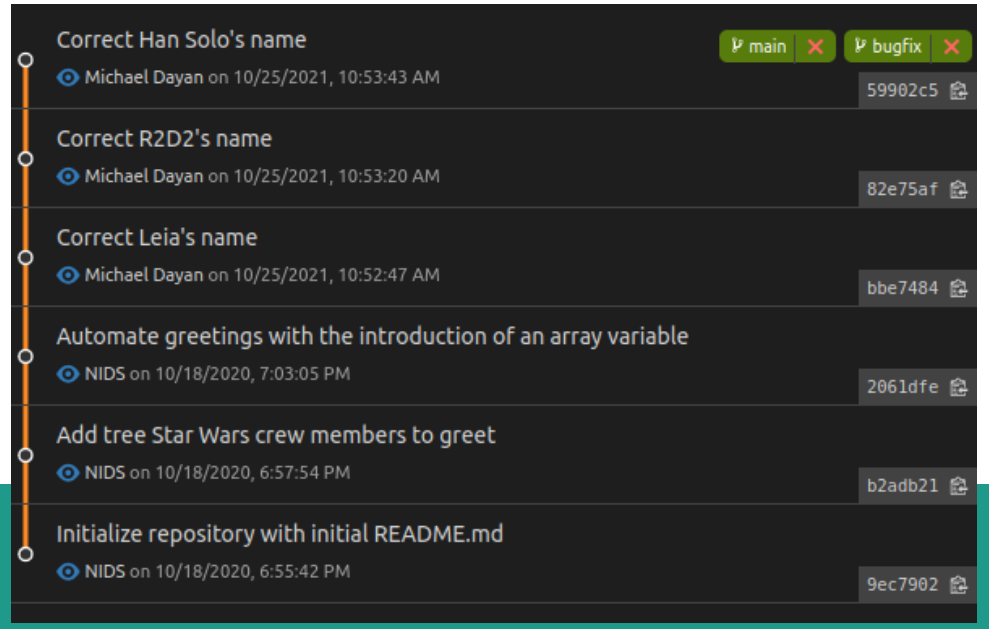
Simplest situation: the topic branch (bugfix in this case) starts from latest commit on main branch

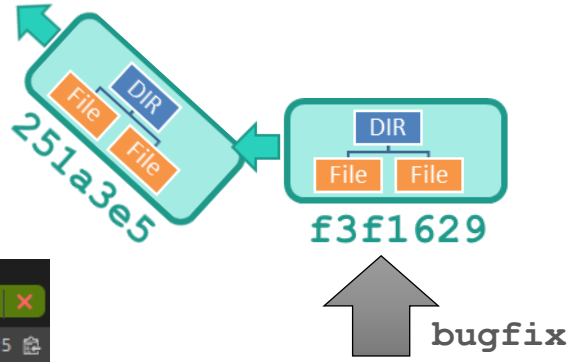
```
git checkout main
git merge bugfix
```





With a fast-forward merge, we cannot tell which commits belonged to the merged branch

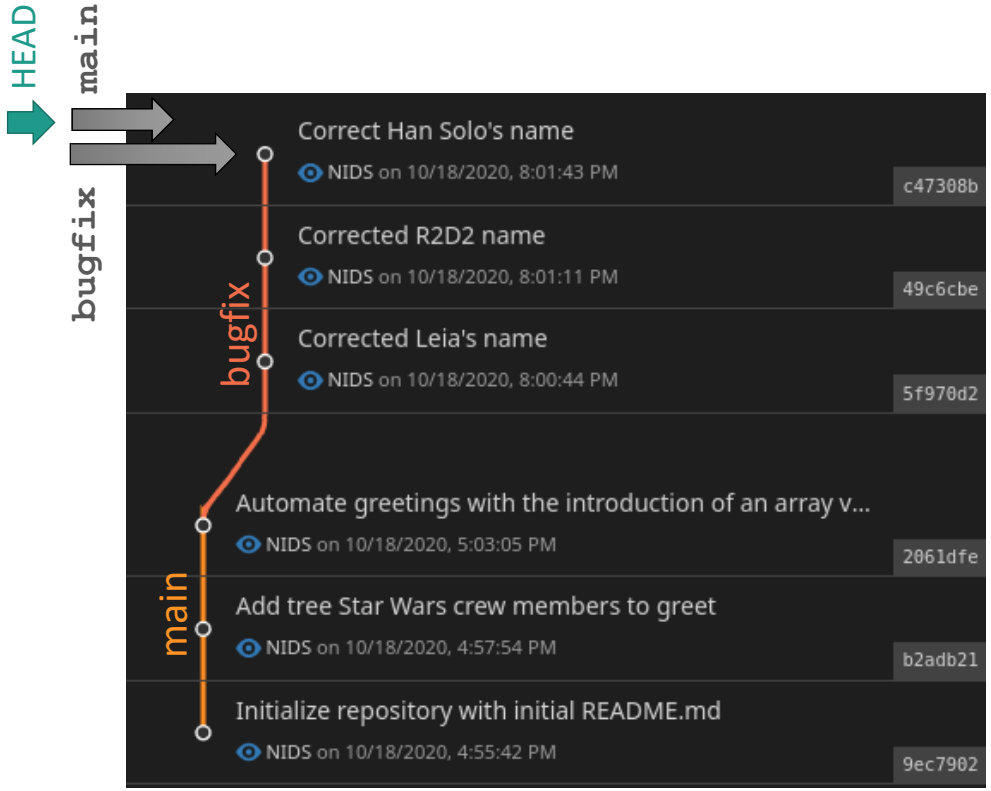


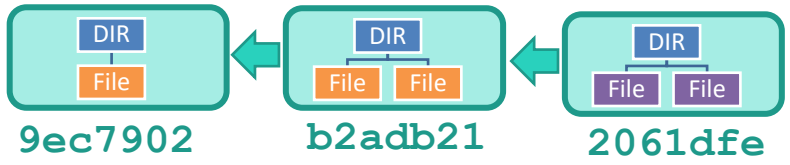


MERGING – FAST-FORWARDING

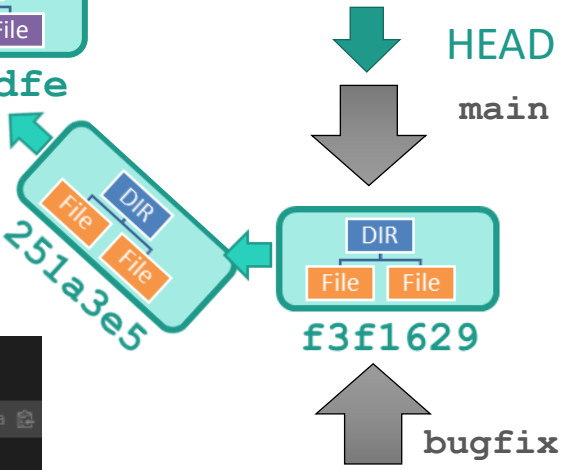
Simplest situation: the topic branch (bugfix in this case) starts from latest commit on main branch

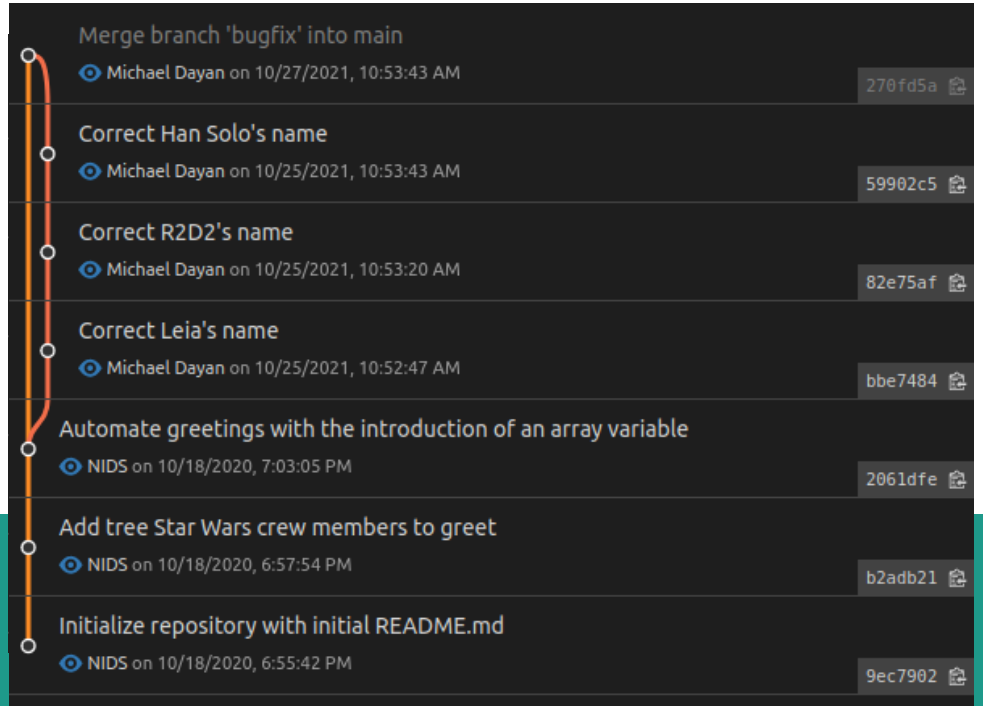
```
git checkout main
git merge bugfix --no-ff
```





An alternative is "no fast-forward" (`--no-ff` option), to keep the history from the merged branch





MERGING – 3-WAY MERGE

More problematic situation: changes also made on main branch

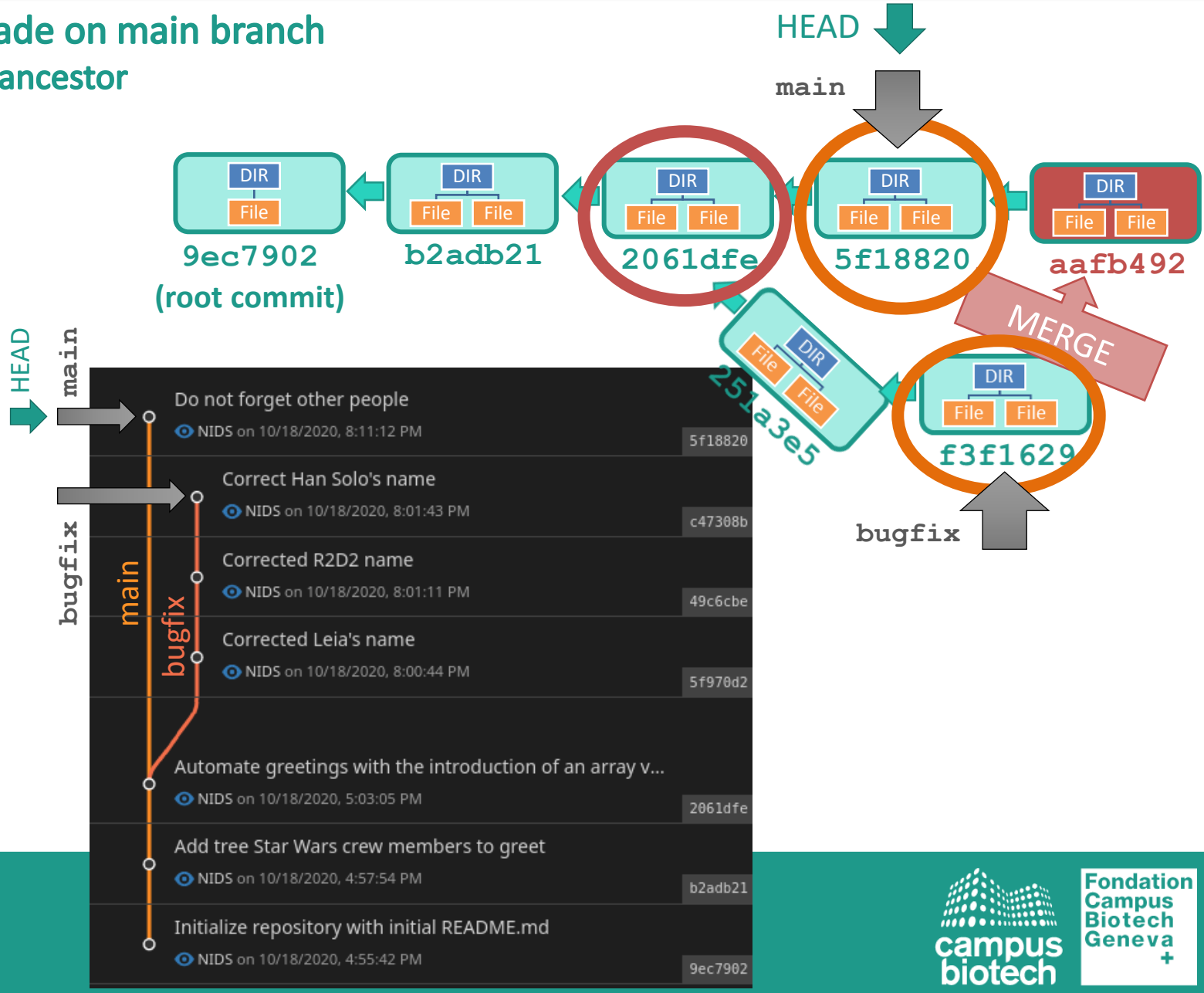
→ both branches have updates from a common ancestor

→ no fast-forward merge possible

This is a 3-way merge:

➢ git examines the tip of the two branches and their common ancestor to do a merge

```
git checkout main
git merge bugfix
```



MERGING – 3-WAY MERGE

More problematic situation: changes also made on main branch

→ both branches have updates from a common ancestor

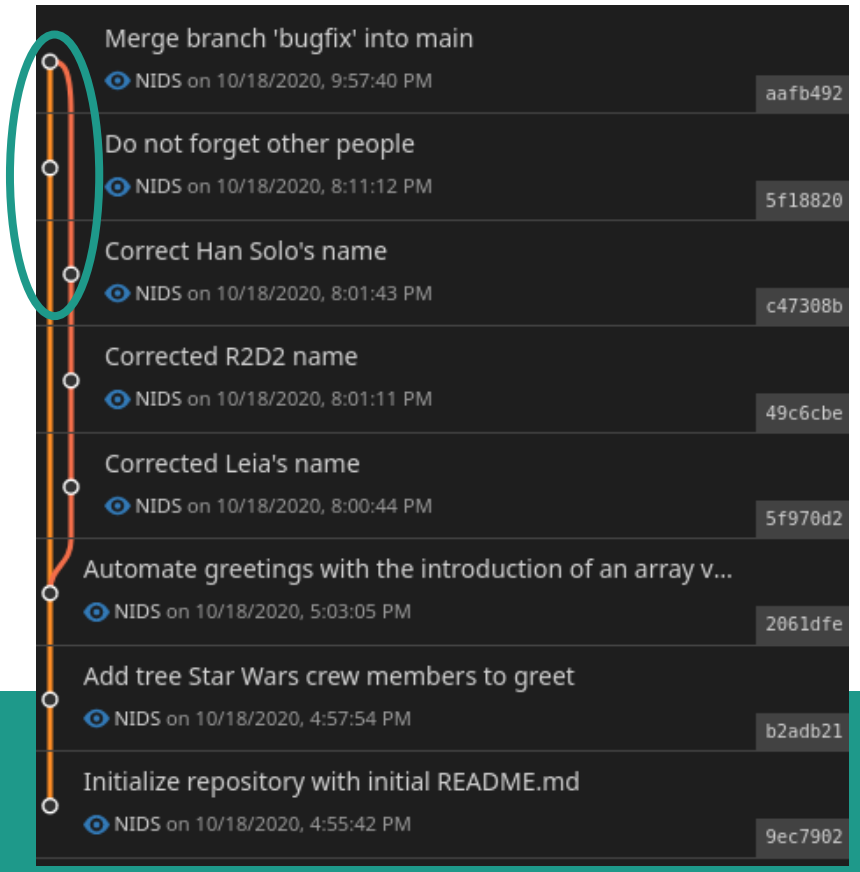
→ no fast-forward merge possible

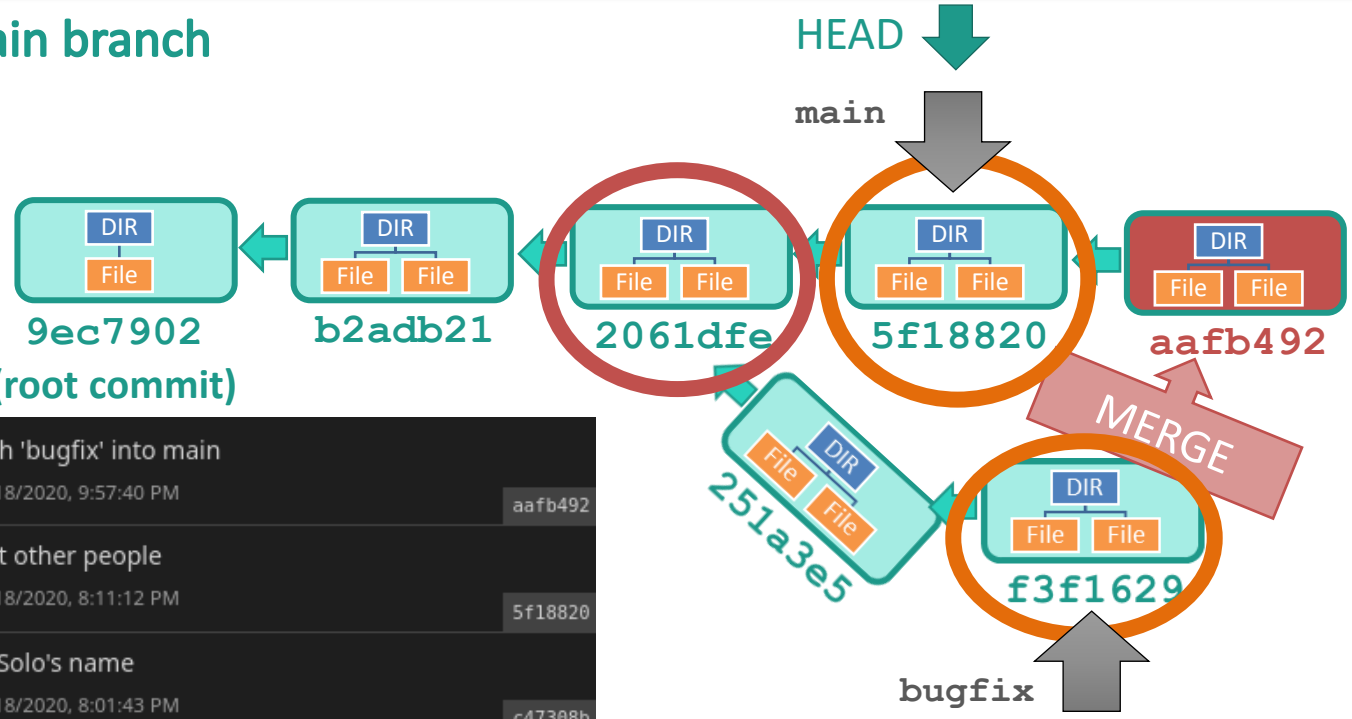
This is a 3-way merge:

➢ git examines the tip of the two branches and their common ancestor to do a merge

```
git checkout main
git merge bugfix
```

➢ this results in a non-linear history





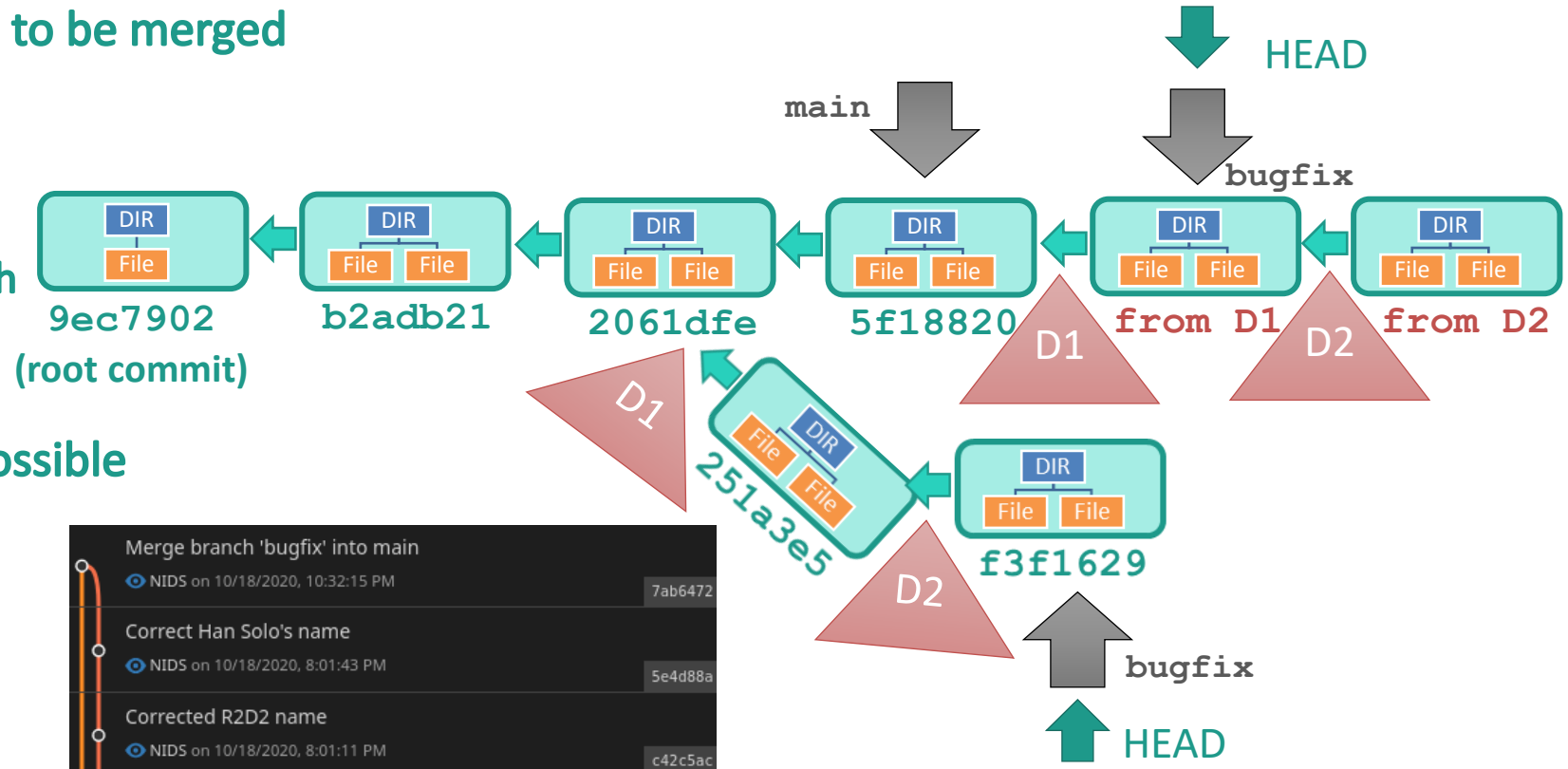
MERGING – REBASE FOR LINEAR GIT HISTORY

Rebase: rewriting history of the branch to be merged

```
git checkout bugfix
git rebase main
```

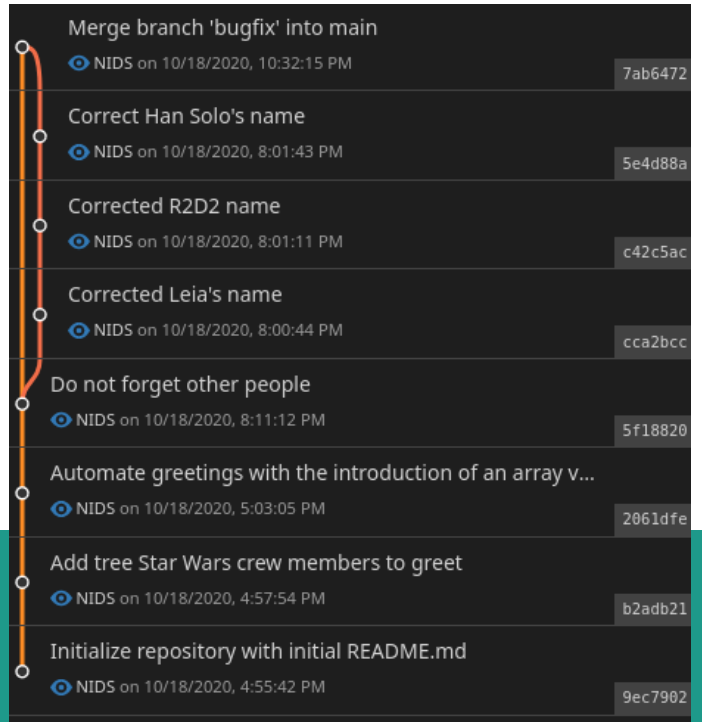
→ compute deltas from `bugfix` branch
→ try applying them in turn to `main` branch

It is the `bugfix` branch which is rebased to the `main` branch, so `bugfix` needs to be activated



The usual fast-forward merge is now possible (use `--no-ff` for cleaner history)

```
git checkout main
git merge bugfix --no-ff
```



If other users work on a branch, this branch history should not be rewritten:

- You should be the only one working on a branch which history is going to be written
- Do not share / push to GitHub branches which may be rebased until you finish rebasing them

MERGING – CHERRY-PICKING

Cherry-picking: pick what you want

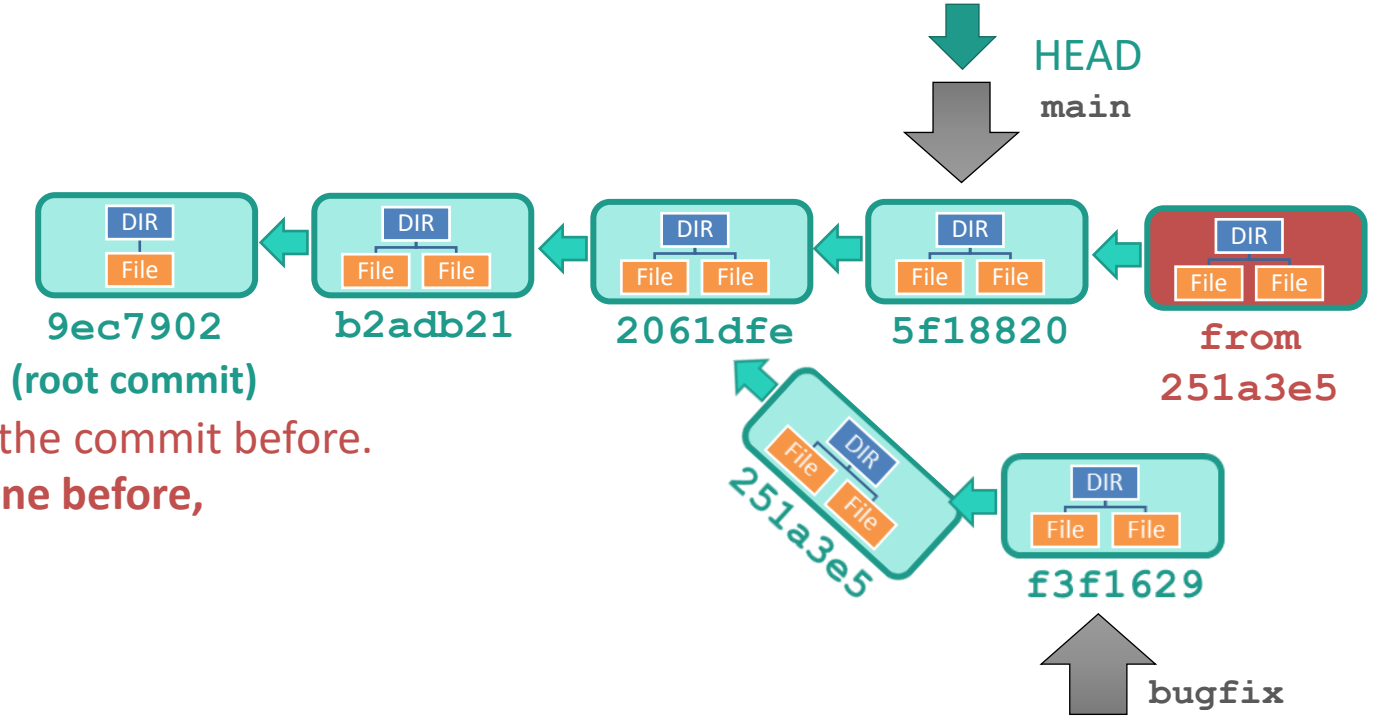
```
git checkout main
```

→ **for a specific commit (e.g. 251a3e5)**

```
git cherry-pick 251a3e5
```

→ **for a range of commit**

```
git cherry-pick 251a3e5~..f3f1629
```



The first commit of the range is ignored, so one can take the commit before.

'~' in front of the commit reference (SHA1) means one before,

'~2' means two before, etc.

GIT LAB 1 – creating a linear history

You are still on laptop 1 and decide to do a rebase of your `bugfix` branch (this is safe since you did not push it to GitHub yet)

- Make sure you are still in `laptop1/sw_greeter` or change to that dir
- Activate the branch `bugfix`
- Visualize graph history in VS Code (click Refresh)
- Rebase the `bugfix` branch onto the `main` branch
- Visualize graph history in VS Code and check the history is now linear
- Activate the branch `main`
- Merge the branch bugfix, with no fast-forward
- Visualize graph in VS Code and check the bugfix commits are visible
- Check git status to print the current status
- Push the changes made to branch `main` to your Github repo
- Activate the branch `bugfix`
- Push to your GitHub remote `origin` the branch `bugfix` and track this remote branch locally.
  TIP: you need to use the –u flag the first time you push a branch not existing on GitHub

```
cd DIR # change to directory DIR (i.e. go inside DIR)
ls # list files and directories in current dir
git checkout BRANCH # activate the branch BRANCH
git branch # list branches (* shows the active branch)
How to rebase branch BRANCH onto the main branch:
  git checkout BRANCH
  git rebase main
git merge BRANCH --no-ff # merge BRANCH in the current
                         # branch without fast-forward
git status # give git status of local environment
git push # push the current branch to the remote
         # GitHub branch it is set to track
git branch -vv # list local branches in very verbose
mode
git checkout BRANCH # activate the branch BRANCH
git push -u ALIAS BRANCH
# push to remote repo ALIAS the branch BRANCH
# and track the remote BRANCH locally (-u option)
```

DEALING WITH GIT ISSUES & MERGE CONFLICTS

At any time you can reset a branch to a given commit with:

```
git reset --hard <commit ID>
```

This command modifies the history so it should be used only on branches not pushed to a remote repository.
Otherwise use `git revert` which applies commits turn by turn to progressively reverse the history:

```
git revert <commit ID>
```

After a merge, git still has a reference to the previous `HEAD`, named `ORIG_HEAD`. So a merge can be cancelled with:

```
git reset --merge ORIG_HEAD
```

The use of tags can be useful in these situations. A tag is an alias (i.e. reference) to a commit.

Two kinds of tag exist:

- simple tags (for local development)

```
    git tag <TAG ALIAS> <commit ID>
e.g. git tag crew3-checkpoint 251a3e5
```

- annotated tag (to share on remote repositories)

```
    git tag -a <TAG ALIAS> <commit ID> -m <message>
e.g. git tag -a v0.1 251a3e5 -m "Star Wars greeter with 3 crew members"
```

When pushing, use the option `--follow-tags` to push annotated tags in the current commits

```
    git push --follow-tags
```

DEALING WITH GIT ISSUES & MERGE CONFLICTS

In case of a conflict between the files to be merged (typically lines at the same position were changed in the commits to be merged), two solutions exist:

- abort commit
- solve conflict

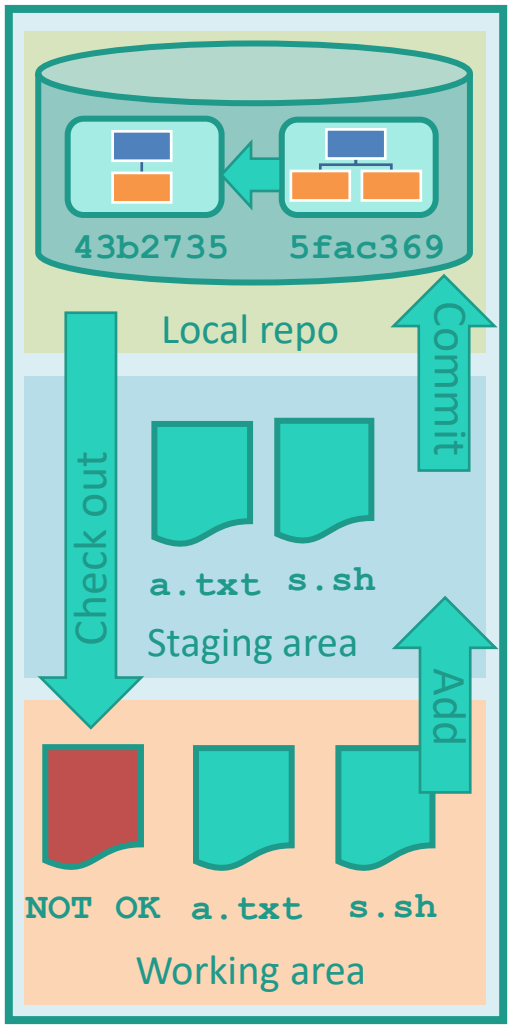
In case of merging conflicts, the abort command depends on the kind of merge:

→ `git merge --abort`
→ `git rebase --abort`
→ `git cherry-pick --abort`

When git displays an error message when the merge didn't succeed:

1) Identify the files with conflicts (git stages the unproblematic files, and leaves the ones with conflicts in the working area)
   → `git status`
2) Modify the files with an editor
   → `git mergetool`
3) Add the corrected files to the staging area
   → `git add .`
4) Commit the staged files
   → `git commit -m MESSAGE`

```
17      public static void main(St
18          // TODO code applicati
        <<<<<<< HEAD
20          // My name is Jeff
        =======
22          // Tyrannosaurus rex
        >>>>>>> origin/master
24      }
```



GIT LAB 2

You switch to your second laptop and decide to add a name to the list of the crew members

Change to the directory `laptop2/starwars_greeter` in your home dir

Edit the file `hello` in that directory (double check you have the right file!) and add another name to the array SW_CREW (e.g. Chewbacca)

Stage, then commit this change.

This will cause a merge conflict when we next pull from the remote. This is because the remote contains the corrected list of names (thanks to bugfix merge) but laptop 2 does not, and it modified the same line of code.

In practice, you should always pull remote branches before doing any modifications. In our case we simulate the case when someone modified the remote while we were working on our files

Try pushing the changes to the remote repo

Pull from the remote repo

Check status with git, then start GUI to edit merge conflict file

Remove the `*.orig` backup files created by the GUI, then commit the corrected file.

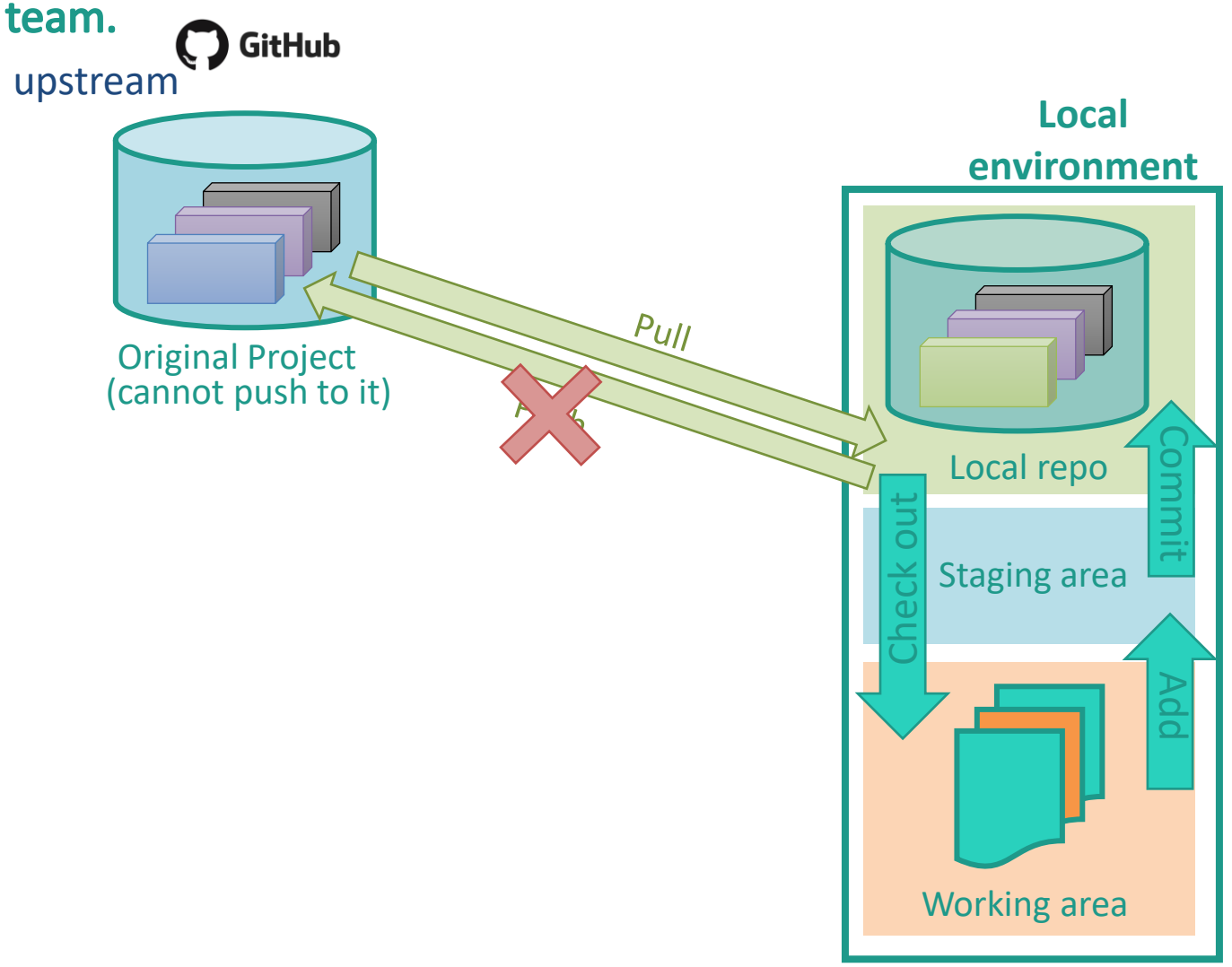
Push to the remote repo and check git status to make sure everything is fine

```
cd DIR # change to directory DIR (i.e. go inside DIR)
ls # list files and directories in current dir
git add FILE # stage FILE
git add . # stage all eligible files
git commit # commit staged files to local repo
           # message will be written via an editor
git commit -m MESSAGE # commit staged files to local
                      # repo with message MESSAGE
git log --oneline # get concise commit history
git push # push the current branch to Github
git pull # pull from the remote to the current branch
git status # give git status of local environment
git mergetool # start GUI to edit merge conflict files
              # - press Return key if asked
              # - edit the middle panel until you
              # are satisfied with the file
              # - click on save button & close the GUI
              # files ending with .orig can be deleted
rm FILE # delete the file FILE
```

COLLABORATING WITH GIT – ON EXTERNAL PROJECTS

You can add "collaborators" on GitHub to code as a team.

But for a remote repository not shared with you (most external projects), **PUSH not possible**
→ use fork & pull model



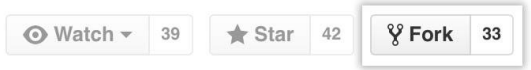
COLLABORATING WITH GIT – ON EXTERNAL PROJECTS

You can add "collaborators" on GitHub to code as a team.

But for a remote repository not shared with you (most external projects), **PUSH not possible**

→ use fork & pull model

- visit the Github repo page and fork the project repo



- this will create a copy in your Github account that you can clone to your computer and do whatever you want with it
- add a remote called `upstream` to be able to get updates from the original project

→ Update your cloned repo from the original project in one step

- get latest `upstream` content

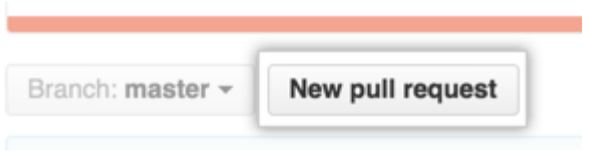
```
git pull upstream main
```

→ Propose changes to the project owner in two steps

- do modifications locally and then push results to `origin`

```
git push
```

- go to the GitHub website and click on "New Pull request"



DONE!

The PR will be sent to the project owner who can decide to include your changes or not

GIT LAB 3

Go to Github and Fork: `https://github.com/IORDS2021/starwars_greeter2`

Create the directory `fork_test` inside your home directory, then go inside `fork_test`

Clone the forked repo to your computer. Make sure to use the SSH location of YOUR repo (i.e. make sure it includes YOUR username)

Go inside the `starwars_greeter2` dir which appeared after cloning

Create the alias `upstream` to point to the SSH location of the repo `starwars_greeter2` of `IORDS2021` (look at code button on GitHub)

List the git remotes to check the alias `origin` (automatically created after cloning) and the alias `upstream` were correctly set.

Pull from `upstream` the branch `main` so that you are up to date

Create a new branch called `feature` and activate it

Add a new character name in the array of the `hello` file

Stage the change, then commit it.

Push your branch to the remote `origin` (i.e. to your forked repo) and track it locally

Go to GitHub and submit a Pull-Request from your `feature` branch to the project owner `main` branch

```
#To fork on GitHub, go to the project URL and
click on the dedicated "Fork" button Fork 33 on the top right

mkdir DIR # create directory DIR
cd DIR # change to directory DIR (i.e. go inside DIR)
git clone git@github.com:<user>/<repo>.git # clone
git remote add ALIAS git@github.com:<user>/<repo>.git
# set alias ALIAS to a Github repository repo owned by user
git remote -v # list all remote aliases
git pull ALIAS BRANCH # pull from the remote ALIAS the
                      # branch BRANCH to the current branch
git branch BRANCH # create the branch BRANCH
git checkout BRANCH # activate the branch BRANCH
git add FILE # stage FILE
git add . # stage all eligible files
git commit # commit files to local repo (launch message editor)
git commit -m MESSAGE # commit staged files to local
                      # repo with message MESSAGE
git log --oneline # get concise commit history
git push -u ALIAS BRANCH # push to remote repo ALIAS the
branch BRANCH and track the remote BRANCH locally (-u option)
Note: Pull-Requests (PR) are done on GitHub from the page
of your forked repo. Look for a button "Pull request" next
to the button "Code". Then click on "Create Pull Request"
```

COURSE SUPPORT

SLACK (iords2021.slack.com)

- Course main channel: `#general`
- Topic channels: `#linux`, `#linux-capstone`, `#git`, `#python`, `#full-example`, `#machine-learning`

→ Check regularly for course info (esp. pinned items)

→ Do not hesitate to ask questions (please reply "in thread")

Week of November 1st :

- Friday 5th, 9AM - 11AM: GIT capstone homework (using GIT for the Linux capstone homework)
- Friday 5th, 11AM - 12PM: 1-to-1 OFFICE HOURS, 20-min slots

→ Book a time slot here: https://tinyurl.com/IORDS-office-hours

EMAIL: methods@fcbg.ch ← Please whitelist!

Delete the SSH key you added to your GitHub repo during the lecture !

HNP METHODS & DATA

campus biotech Fondation Campus Biotech Geneva


Thank You!
Michael Dayan: methods@fcbg.ch
H N P
METHODS & DATA
campus biotech
Fondation Campus Biotech Geneva