TÉCNICO LISBOA

Circuit Theory and Electronics Fundamentals

Lecture 5: Introduction to the lab classes online – 1st Part

- Why Linux
- Linux terminal and bash
- Git
- The TCFE git repository

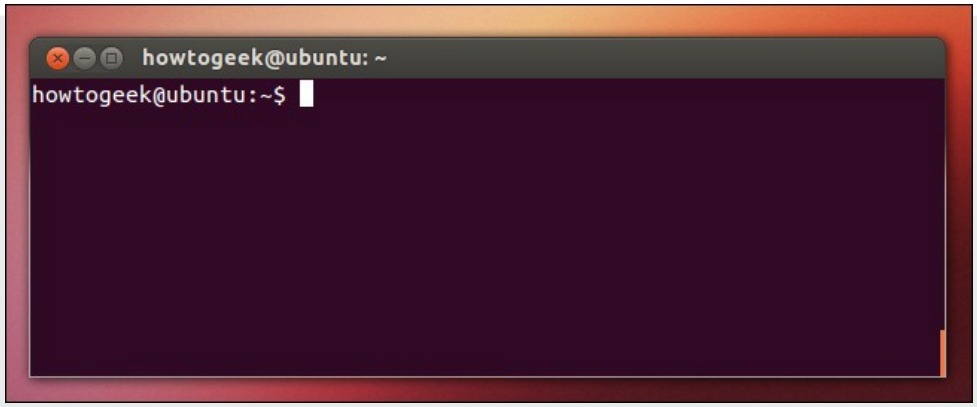
Doing electronics is no longer a garage affair: it is a desktop affair!

- Use suitable tools!

Why Linux and not Windows

1) Linux is better for programming, with many good tools available for free
2) Reliability: almost never breaks or needs rebooting, and runs very stably
3) Excellent community support: you are always one Google search away from finding the answer you need
4) Security: not invulnerable but does not catch common viruses and malware
5) Privacy: not invulnerable but much more unlikely to get spyware

The Linux Terminal and Bash

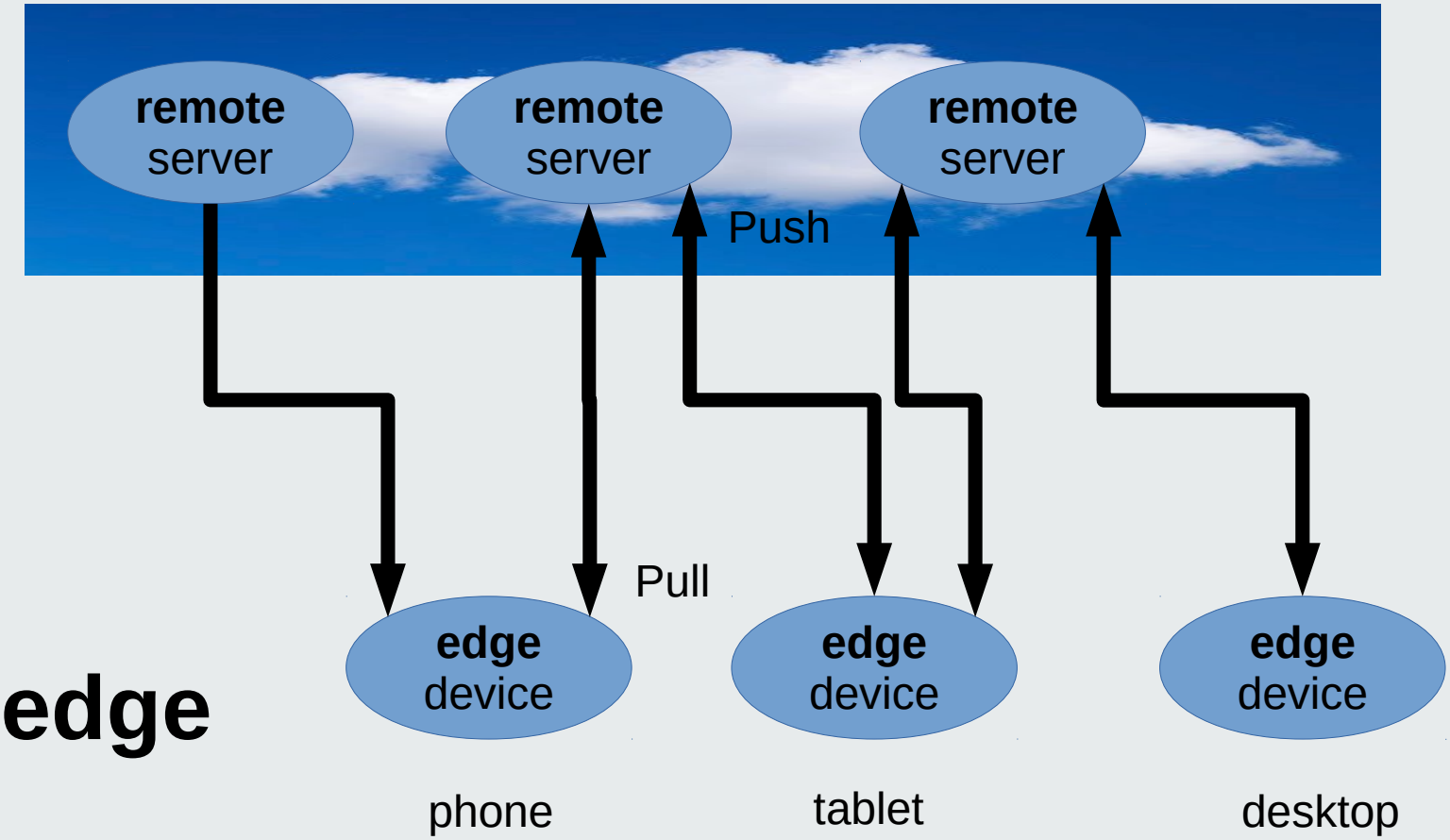


- Although most Linux distros come with a Graphical User Interface (GUI), the Command Line Interface (CLI) is the preferred interaction way
- The CLI is a Linux Terminal running a command interpreter called ***bash***
- Bash provides and environment (shell) for running installed programs, and Linux comes with an infinity of excellent ***open-source*** programs...
- Bash is also a full-fledged programming language with assignments, conditions and loops: long sequences of bash commands can be put in bash scripts and run in a fully automatic way!
- Humans can type a lot faster than they can select, click and drag; if you remember a program's options by force of repetition, you don't want to go through long menu chains every time you need to run it – you prefer to type a command and few options or script it

TÉCNICO LISBOA

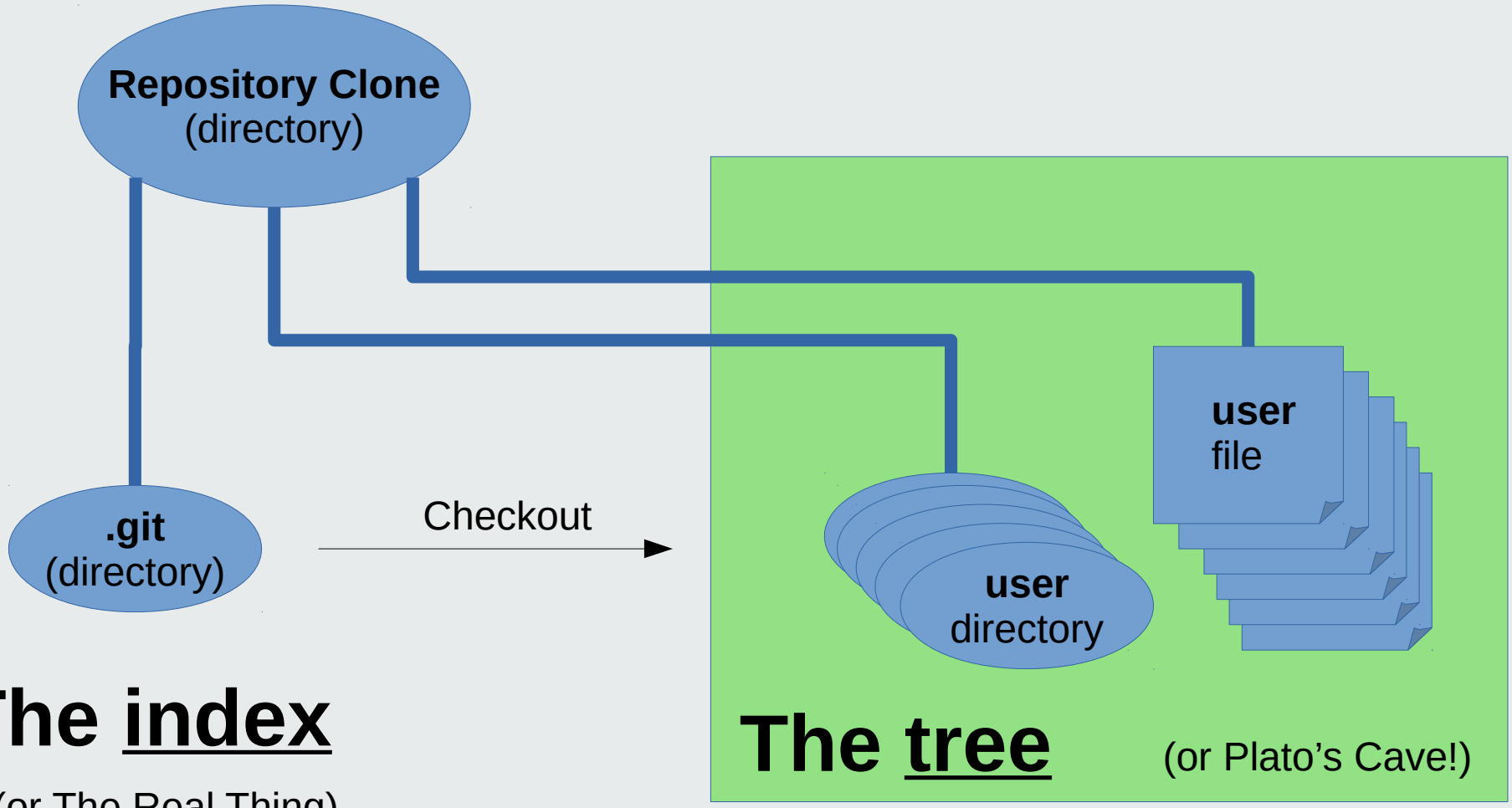
- Project management and version control tool
- Successor of CVS and SVN, but much better
- Developed by Linus Torvalds (also Linux creator)
- Managing your project files efficiently
- Reduce storage and backup needs
- Manage different versions and development branches in large teams
- Access the project files anywhere

The cloud



The edge

At the Edge



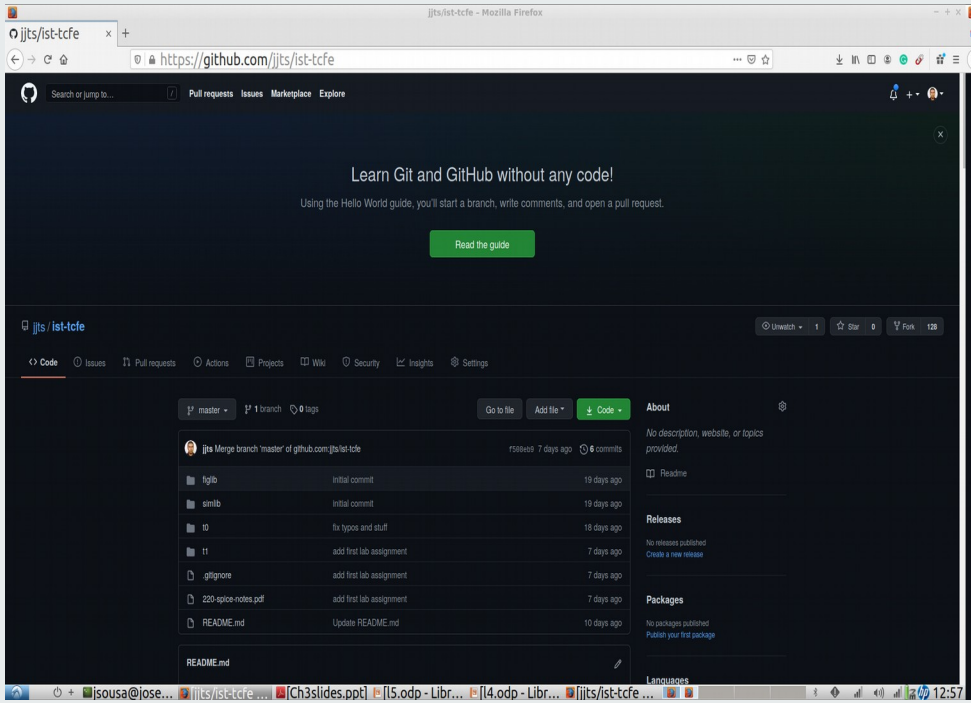
The index

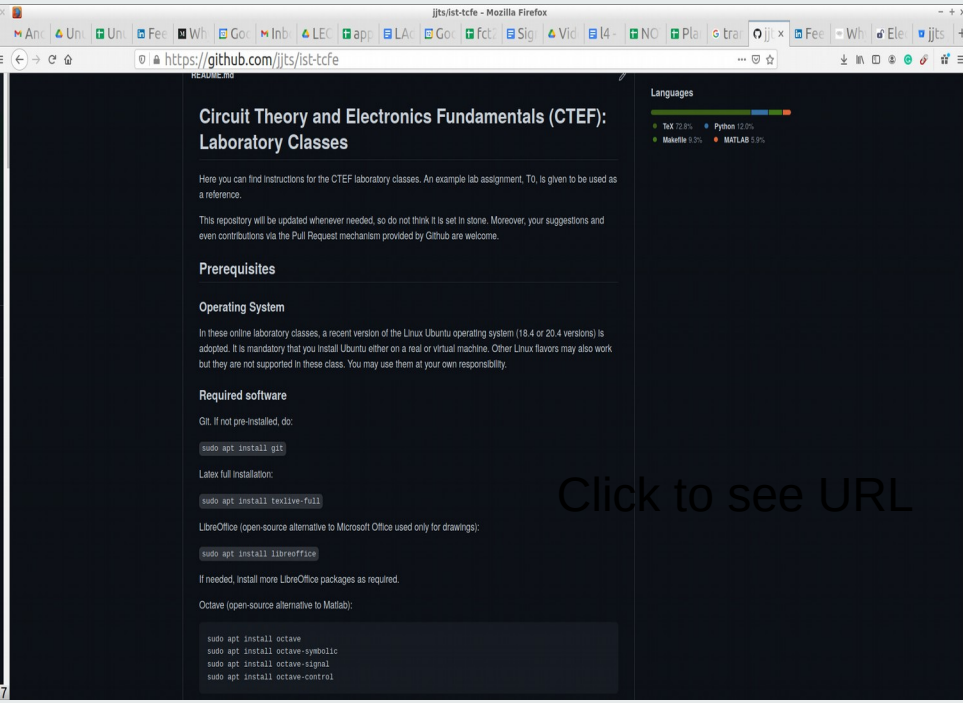
(or The Real Thing)

The ist-tcfe git repository

- Hosted by Github (*Microsoft*)
- Read the README file
- Fork the repository
- Clone the forked repository
- Move into its directory
- Move into the t0 directory to issue the “make” command to
  - Run the theoretical analysis script
  - Run the circuit simulation script
  - Run Latex to produce an automatically generated project report

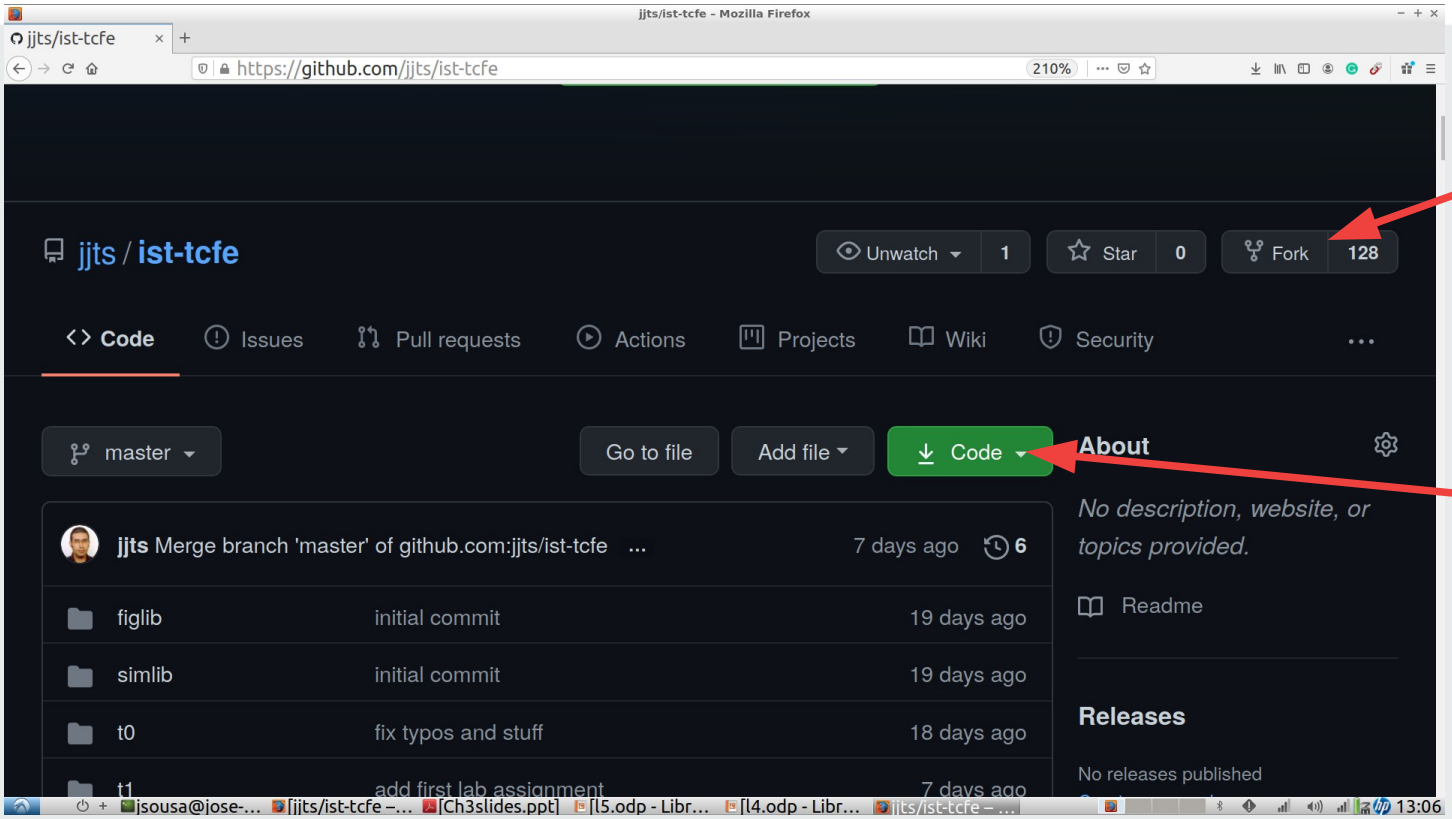
TCFE at Github





- Hosted by Github
- Read the README file
- https://github.com/jjts/ist-tcfe

Fork the ist-tcfe repository



Click to *fork*

Click to see URL

- *Forking* the repo will create a remote repository copy for you
- Forks can send “Pull Requests” to the original repo
  - New additions or error corrections. Fork owner becomes a *contributor*

Git set up on local machine

- Use Ubuntu 18.04 or 20.04 LTS
- Git most likely pre-installed; if not:

  sudo apt install git
- Tell Git who you are. It's mandatory:

  git config –global user.name "Your Name"

  git config –global user.email "your@email.com"

Clone your fork

- Clone YOUR FORK in your local machine!
- Do not clone the original repository!

  git clone https://github.com/you/ist-tcfe
- Creates a copy of your repository (the fork) in your local machine
- Now you can edit and create files to complete the lab assignments

Check the *Status* of your repo

- Once you have modified folders and files you may want to check the status of your repository

  git status

  – tells you which files are new, deleted or modified as well as other relevant status info
- It is git's most used command but you will only understand why when you start using git

Commit your work

- Once you have advanced your work you may want to take a snapshot of it by creating a ***commit***
- A commit is not the final work, not even an intermediate milestone!
- It is just some advance, big or small: a point in history you may need to return in the future
- A rich commit history makes it is easy to go back to a previous commit and fix an issue

Stage your work

- Before committing you need to ***stage*** your work using

  git add <dir>

  – Stages all files and folders in folder <dir> and below; example:

  git add .

  – Stages all files and folders in current directory and below
- Staging gives you the last chance to look at the stuff that will be committed
- Do not stage and commit generated files! Octave, ngspice and Latex may generate lots of files that can safely be deleted; they are always generated when these tools run. These files should be deleted with “make clean” or listed in a file called .gitignore that is placed in the root directory.

git commit

- Once you've staged all the files you want to commit it is time to create the commit:

  git commit -m"your commit message"

- Commit messages should, in a single line, describe the commit. If you do simply:

  git commit

  a text editor will open for you to enter the commit message in the 1st line, followed by one blank line, followed by the commit text, usually a paragraph describing the commit

- Each commit is identified by a ***hash***, for example:

  45fb228ee6ee1f4d3def8605d2ca068f51c3d558

Checking commit history

- Git allows you to check **who** did **what** and **when** with the command

  git log

- A simplified log is presented with

  git log --oneline

- It is a very useful command but you can only appreciate it after you start using git

Saving and sharing your work on Github

- Once you commit your work, the command

  git status

  should tell you that there is no more work to commit and that your clone is ahead of your remote
- To save and share your work on Github (or another platform of your choice), use the command

  git push
- You can control who you share the work with, your group colleagues and the instructor

Downloading updates to your fork

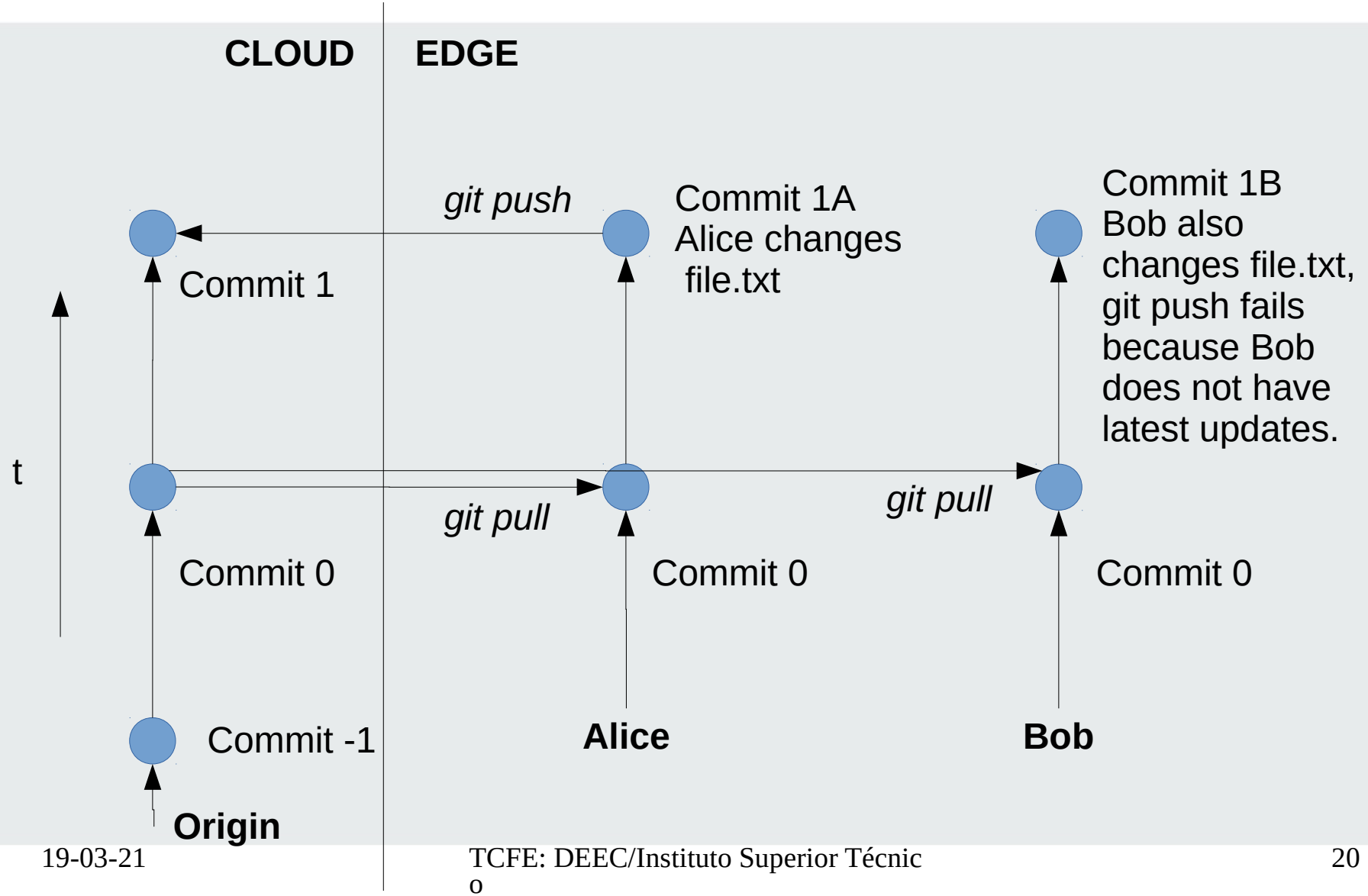
- To download updates that may have been *pushed upstream* by one of your colleagues use the command

  git pull

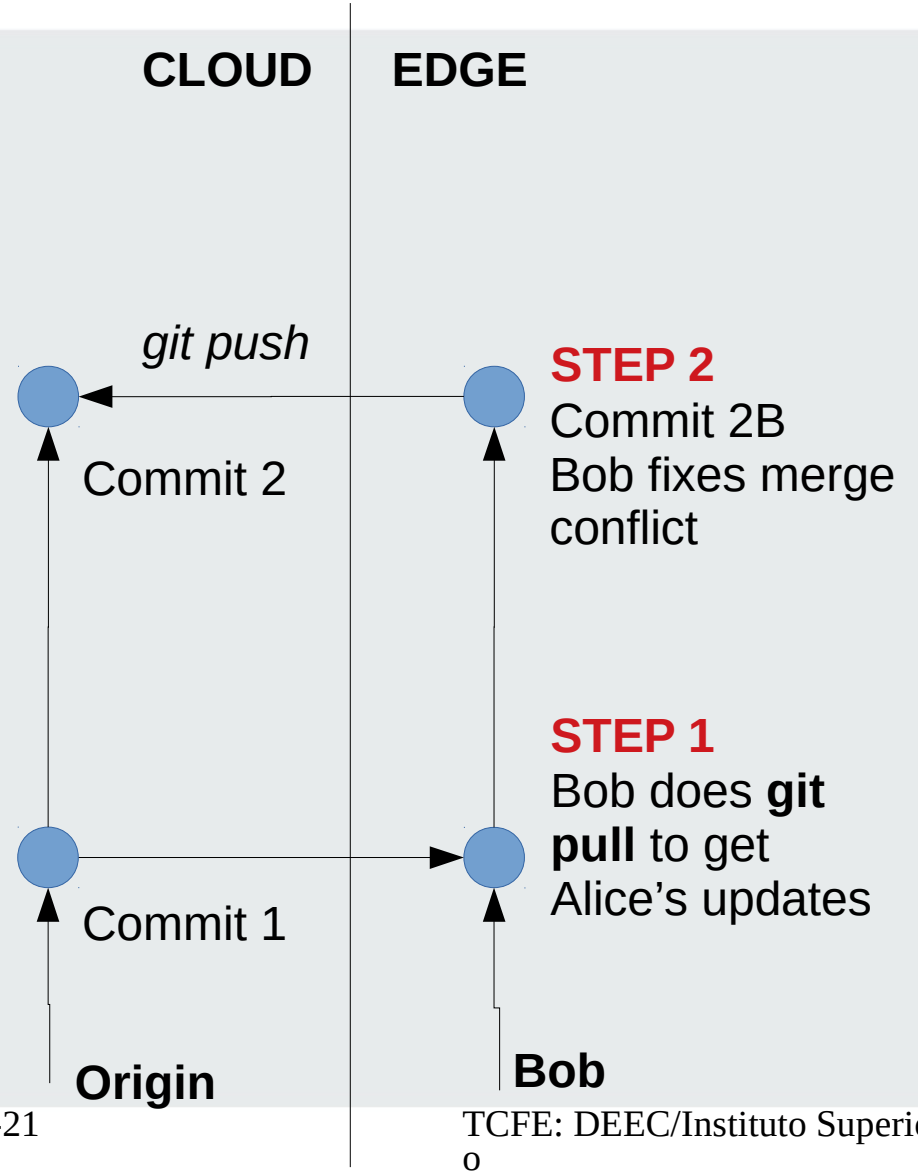
  It will synchronise your clone with the default ***remote*** (your fork) called ***origin***

  The git pull command can either finish cleanly or you may get a ***merge conflict***

Merge conflict



Solving merge conflict



STEP 1 SCENARIOS

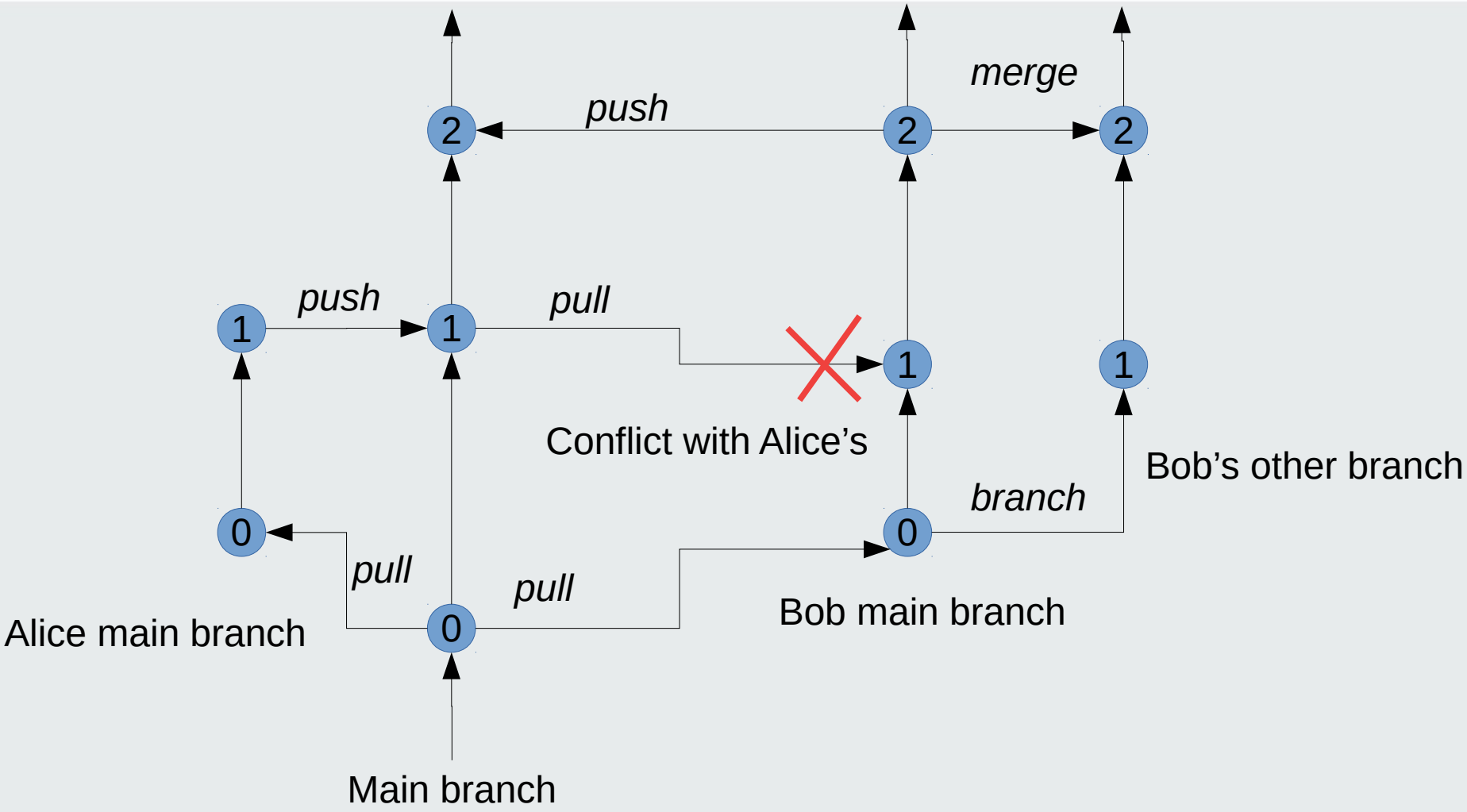
Scenario 1: Git manages to merge the remote main branch and Bob's main branch automatically.

Scenario 2: Git can't merge, Bob gets ***merge conflict***! Both Alice and Bob have changed the same lines in file.txt, Git marks them:

<<<<<<< HEAD:file.txt
Hello world
=======
Goodbye
>>>>>>> 77976da...:file.txt

Bob needs to edit this file, choose what to keep, delete the markers and commit the changes. This fixes the merge conflict.

Development branches Graph



Downloading updates from the instructor's main branch

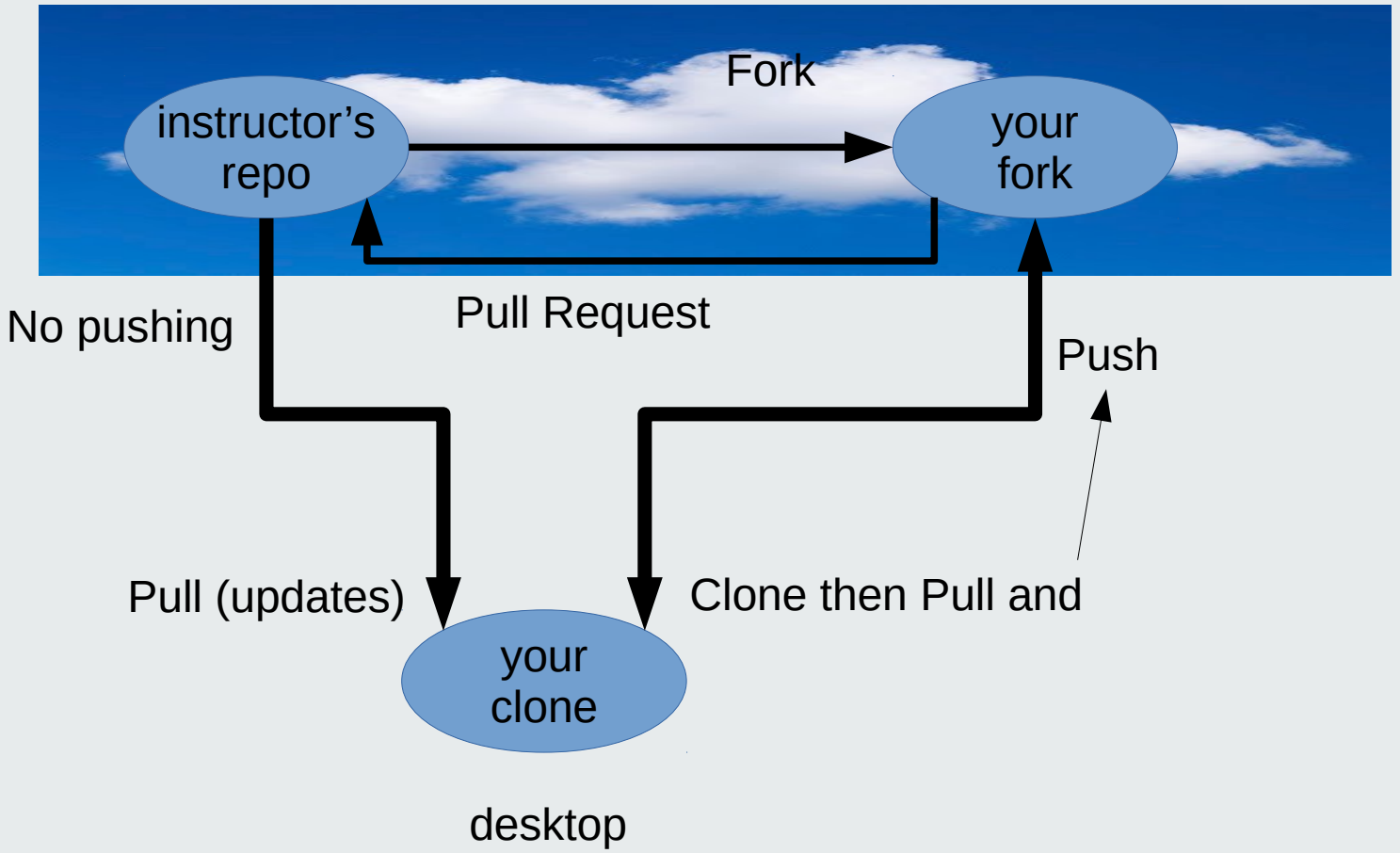
- The instructor has added new material after you have forked his repository
- How can you get new updates, for example, the newest lab assignment?
- Short answer: add his repo as a second remote to your clone and pull down his updates
- List your remotes: (you may have as many as you need)

git remote -v

origin https://github.com:you/ist-tcfe-staff.git (fetch)

origin https://github.com:you/ist-tcfe-staff.git (push)

Working with two remotes



Adding the instructor's repo as a remote

- Add the instructor's remote

  git remote add tcfe https://github.com/jjts/ist-tcfe.git
- List remotes again

  git remote -v

  origin https://github.com:you/ist-tcfe-staff.git (fetch)

  origin https://github.com:you/ist-tcfe-staff.git (push)

  tcfe https://github.com:jjts/ist-tcfe-staff.git (fetch)

  tcfe https://github.com:jjts/ist-tcfe-staff.git (push)

Pulling the instructor's repo updates

- Pull updates from the instructor's remote

  git pull tcfe master
- It pulls the branch *master* from remote repo *tcfe* and merges it into your main branch
- It will merge the latest updates from the instructor's main branch into your clone's main branch
- Merge conflicts are not expected since the instructor only adds or modifies instruction files, not work files

Git command summary

- **git config --**global user.name “Your Name”
- **git config --**global user.email “your@fantastic.email”
- **git status**: shows difference between index and tree
- **git add**: stages the files to be commited in index
- **git commit [-m”message”]**: creates a new development node (commit or hash)
- **git push**: sends your commits to a remote
- **git branch**: creates a development branch
- **git checkout**: checks out branch from index to tree
- **git log:** shows commit history
- **git show**: shows contents in any commit
- **git diff <hash> [--name-status]:** shows difference to a commit hash
- **git fetch [*remote branch*]**: downloads branches from a remote
- **git merge [*branch*]: merges *branch* into current branch**
- **git pull**: same as **git fetch + git merge**
- **git remote [-v | add | remove | rename]:** shows, adds, removes or renames remote servers