Software Project Management - Laboratory
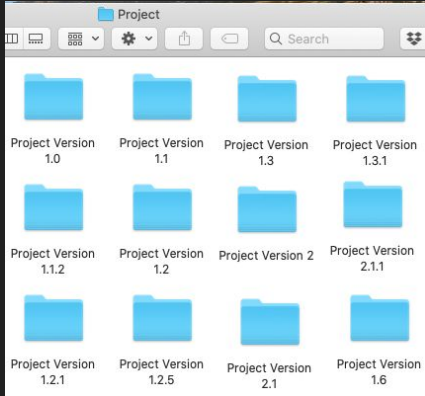
Lecture n° 3
A.Y. 2020-2021

Prof. Fabrizio Fornari

Version Control

A version control system records changes to a file or set of files over time so that you can recall specific versions.

How to do it?

Manually

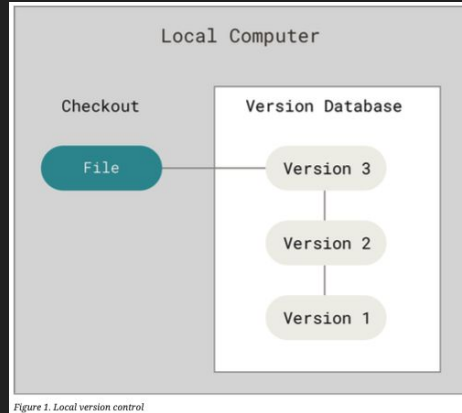## Local Version Control Systems

Figure 1. Local version control

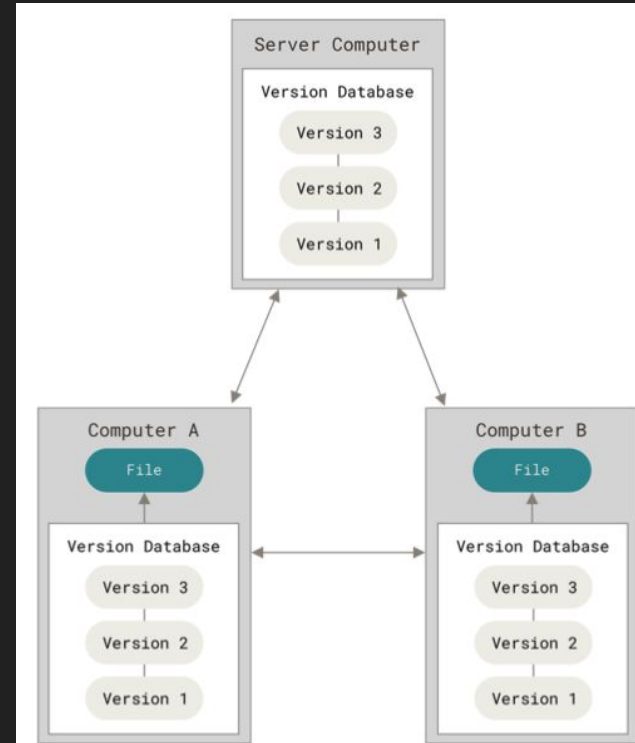## Distributed Version Control Systems

Figure 3. Distributed version control

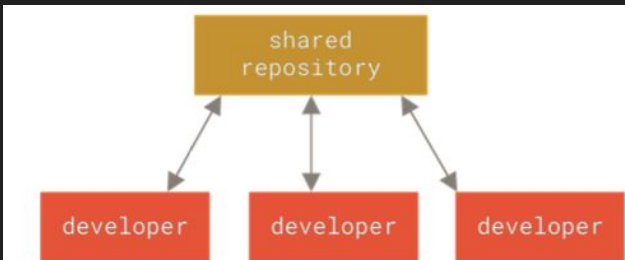## Centralized Version Control Systems

Figure 2. Centralized version control

Distributed Version Control System

Distributed means that there is no main server and all of the full history of the project is available once you cloned the project.
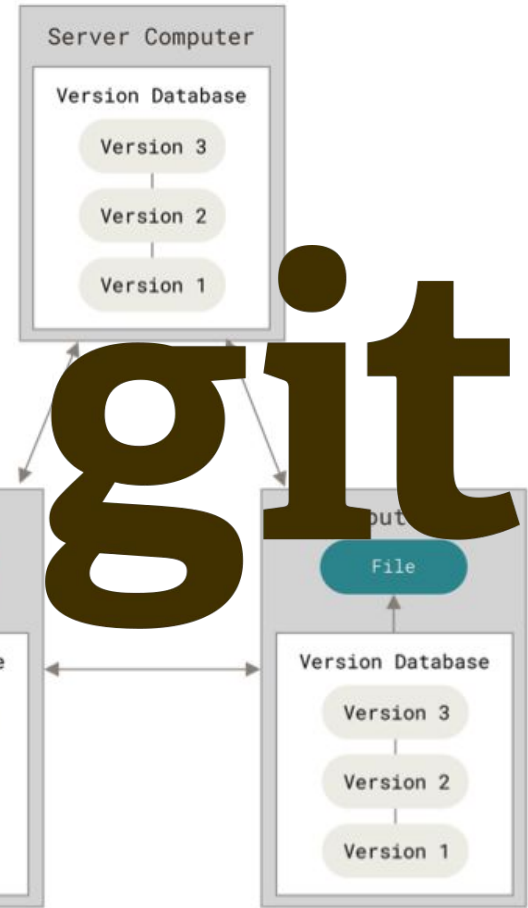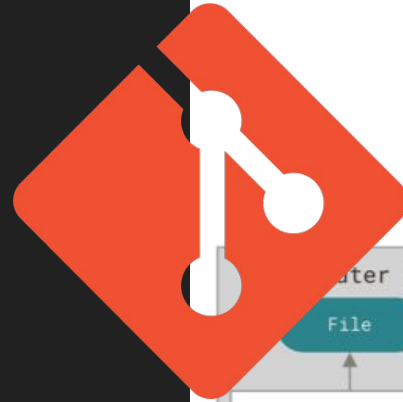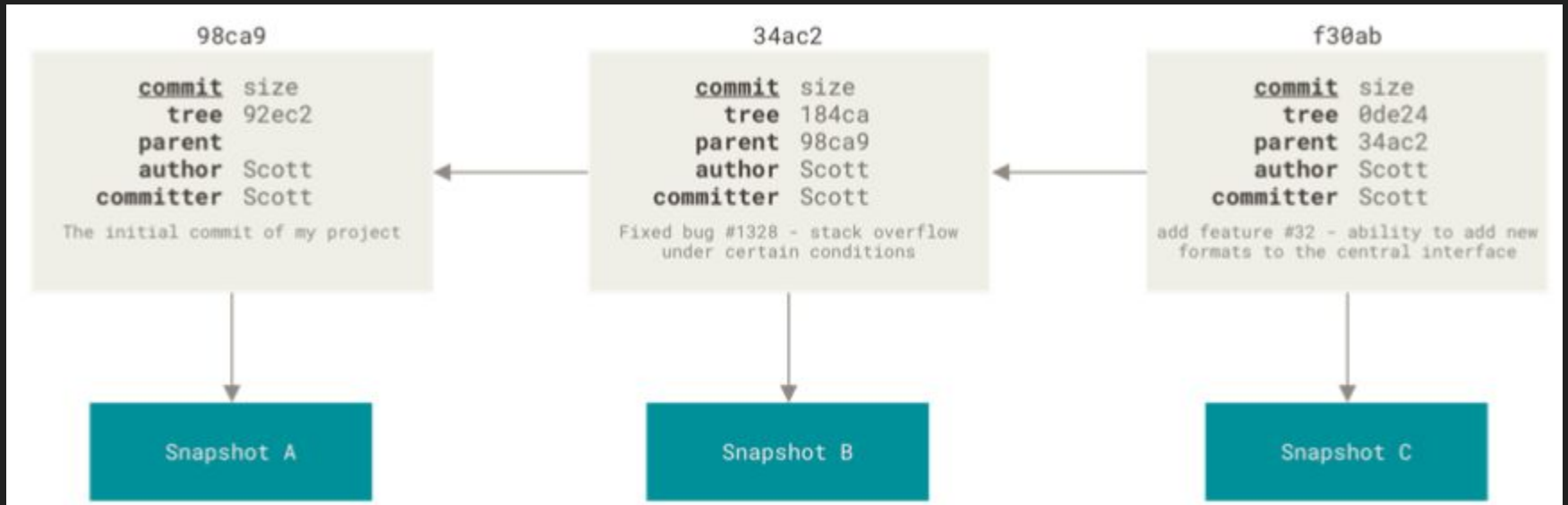

Figure 3. Distributed version control

Set up a Repository

1. Create a new folder and open a terminal in that folder. $ cd pathToTheFolder/FolderName

2. Initialize a repository$ git init

3. Create one file and commit,

4. Create a second file and commit,

5. Create a third file and commit

Commits

Commits can be seen as a list where each commit points to his parent.

Branching

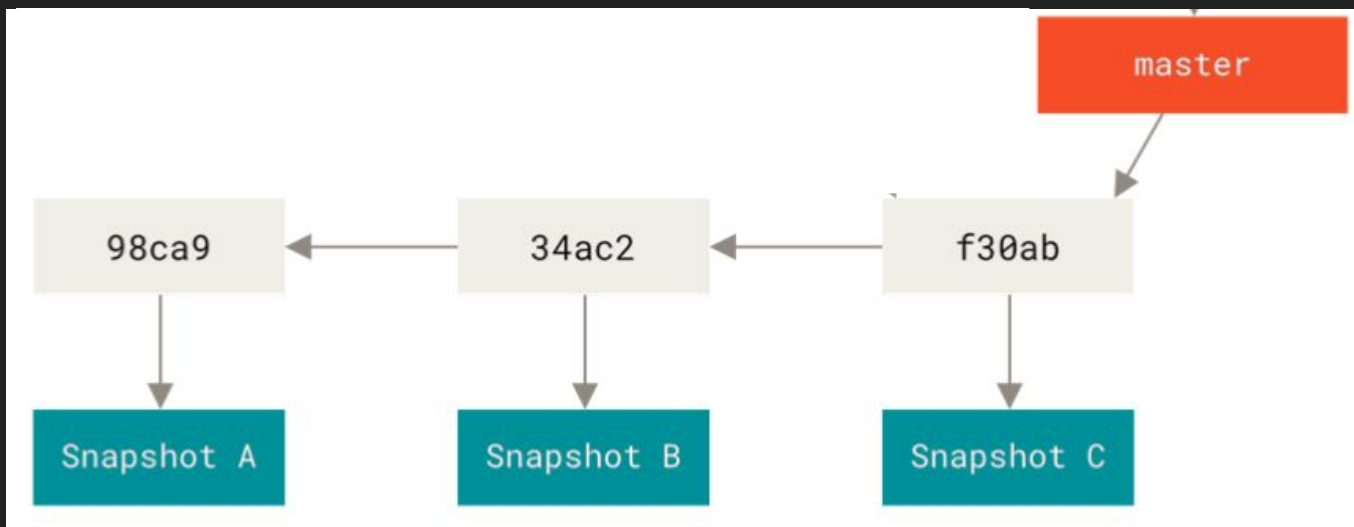Branching is one of the major characteristics of Git.

A git branch can be seen as a pointer that can point to any of the available commits.

The default branch name in Git is *master*.

Branching

6. Run git log --oneline --decorate

```
(base) fabriziounicam:MySecondRepo user$ git log --oneline --decorate
bbaf42a (HEAD -> master) Added a Third File
bf95483 Added a Second File
4a757df Added a First File
```
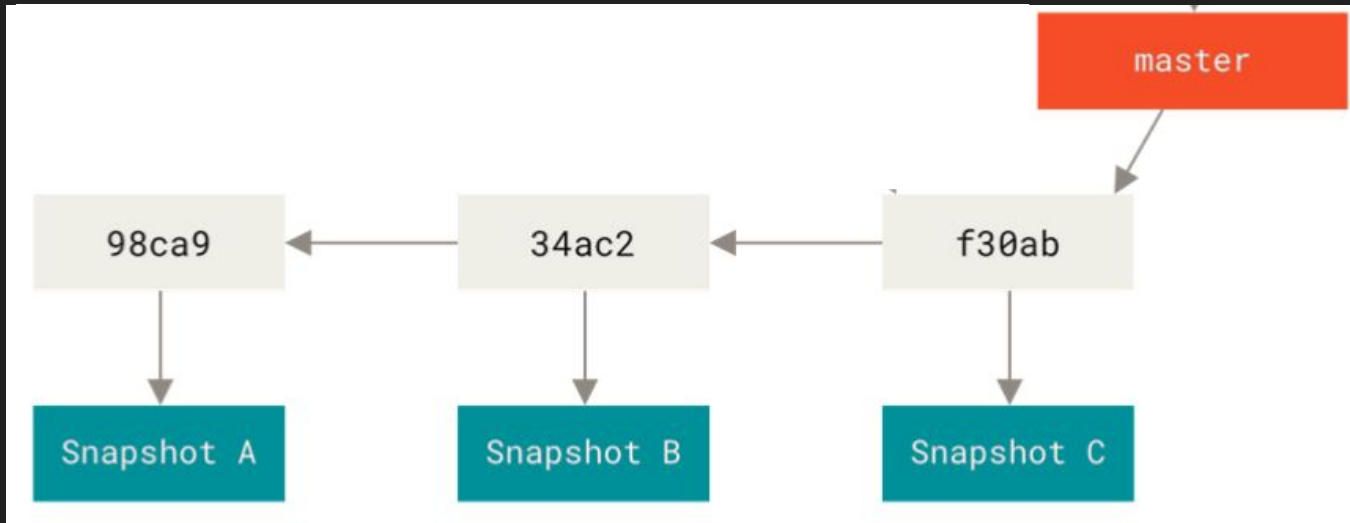
Creating a New Branch

1. Run git branch testing
2. Run git status
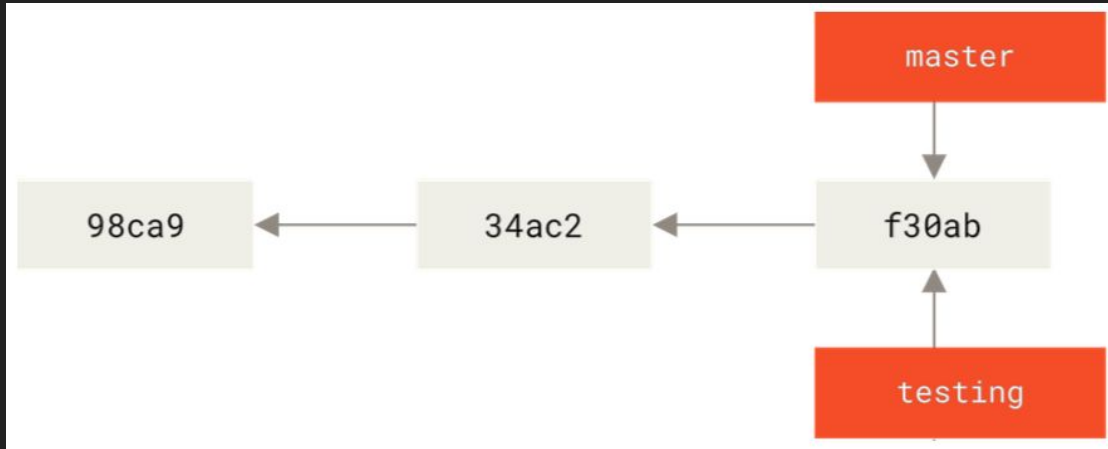
The git branch command only created a new branch — it didn't switch to that.

3. Run git branch -a

```
[fabriziounicam:Local user$ git branch -a
* master
  testing
```

Creating a New Branch

1. Run git log --oneline --decorate

```
[fabriziounicam:Local user$ git log --oneline --decorate
d0d6b7d (HEAD -> master, testing) added .gitignore file
dd11a39 added a FifthFile and a SixthFile
ca9da00 deleted the fourth file
da1b4ea added a fourth file
d087029 Added one line to ThirdFile
```

Switching to a New Branch

2. Run git checkout testing

```
fabriziounicam:Local user$ git checkout testing
Switched to branch 'testing'
```

This moves HEAD to point to the testing branch.

Commit to a New Branch

3. Create a new file (or do some changes to the already available files)
4. Commit those changes
5. Run git log --oneline --decorate --graph --all

```
(base) fabriziounicam:MySecondRepo user$ git log --oneline --decorate --graph --all
* f1a819b (HEAD -> testing) Added a Fourth File
* bbaf42a (master) Added a Third File
* bf95483 Added a Second File
* 4a757df Added a First File
```

Your testing branch has moved forward
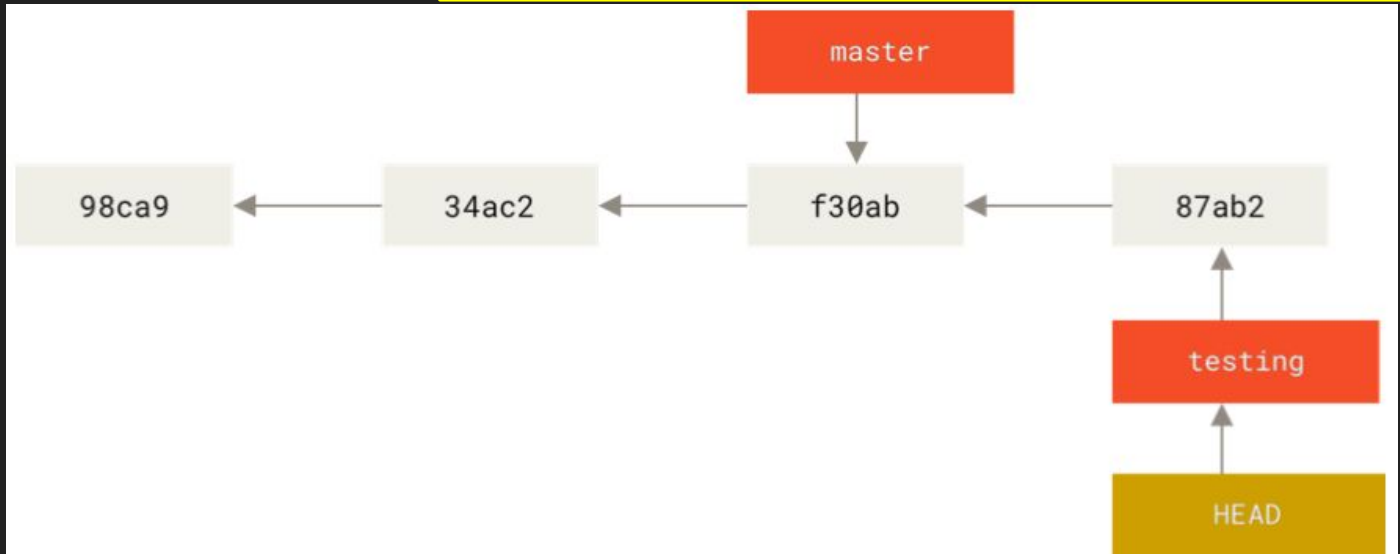
Back to master branch

6. Switch back to the master branch
7. Run git log --oneline --decorate --graph --all

```
(base) fabriziounicam:MySecondRepo user$ git log --oneline --decorate --graph --all
* f1a819b (testing) Added a Fourth File
* bbaf42a (HEAD -> master) Added a Third File
* bf95483 Added a Second File
* 4a757df Added a First File
```
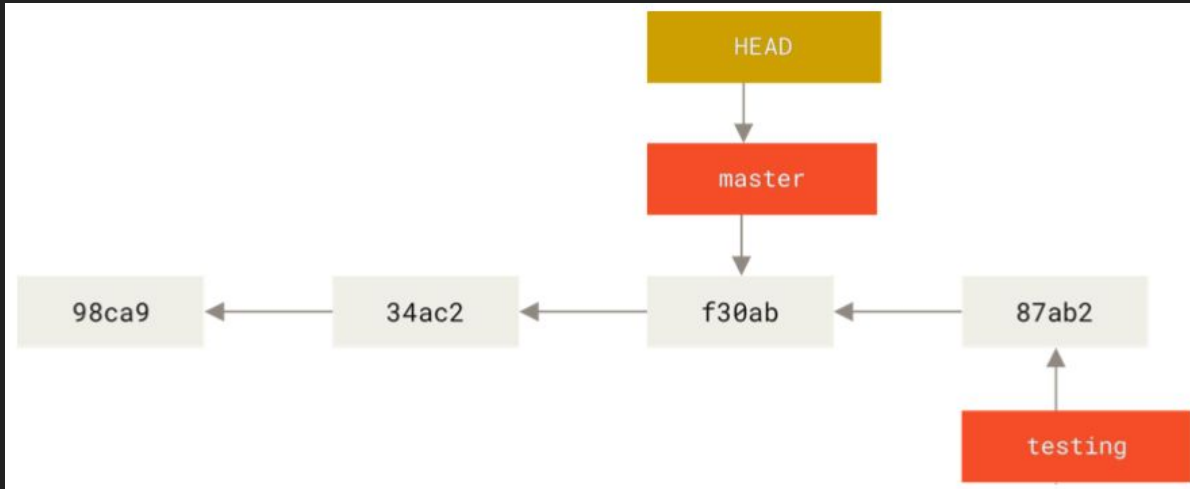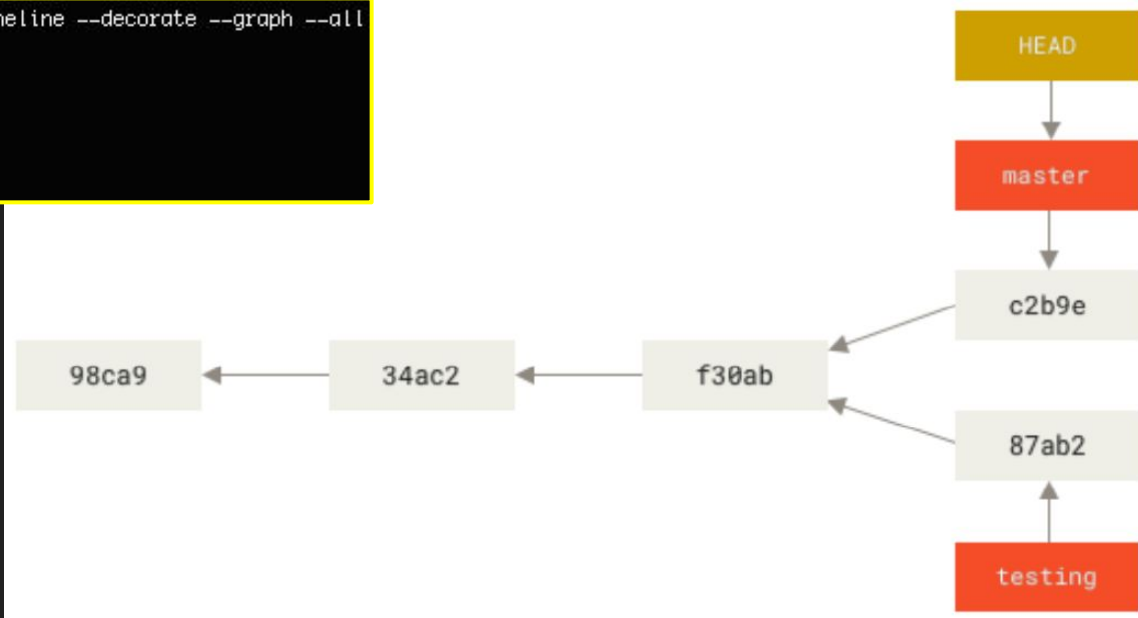
- HEAD pointer is back to point to the master branch.
- Files in your working directory are back to the snapshot that master points to.

Commit to master branch

8.   Make some changes to a file, or create a new one, or remove one.
9.   Commit those changes
10. Run git log --oneline --decorate --graph --all

Branching and Merging

Let's go through a simple example of branching and merging with a workflow that you might use in the real world. You'll follow these steps:

1. Do some work on a website.
2. Create a branch for a new user story you're working on.
3. Do some work in that branch.

At this stage, you'll receive a call that another issue is critical and you need a hotfix. You'll do the following:

1. Switch to your production branch.
2. Create a branch to add the hotfix.
3. After it's tested, merge the hotfix branch, and push to production.
4. Switch back to your original user story and continue working.

Branching and Merging

Let's say you're working on your project and have a couple of commits already on the master branch.

Now you decide to focus on a specific issue; imagine it as a user story that you want to add or something you want to fix.

1. Run git checkout -b *IssueNumber*

    A short way for: git branch *IssueNumber*
                                  git checkout *IssueNumber*
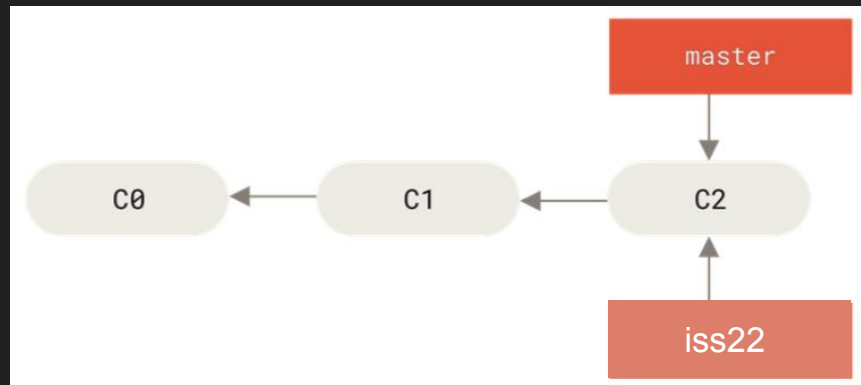
```
fabriziounicam:Local user$ git checkout -b iss22
Switched to a new branch 'iss22'
```

Branching and Merging

You do some changes and you commit

```
fabriziounicam:Local user$ vi index.html
fabriziounicam:Local user$ git commit -a -m 'Create new footer [issue 22]'
[iss22 a44da98] Create new footer [issue 22]
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Now you get the call that there is an issue with the website, and you need to fix it immediately.

What do you do?

Branching and Merging

You do some changes and you commit

```
[fabriziounicam:Local user$ vi index.html
[fabriziounicam:Local user$ git commit -a -m 'Create new footer [issue 22]'
[iss22 a44da98] Create new footer [issue 22]
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Now you get the call that there is an issue with the website, and you need to fix it immediately.

1. Run git checkout master

2. You have a hotfix to make. Let's create a hotfix branch on which to work until it's completed.

3. Run git checkout -b hotfix

4. Modify index.html file and commit the changes

Branching and Merging
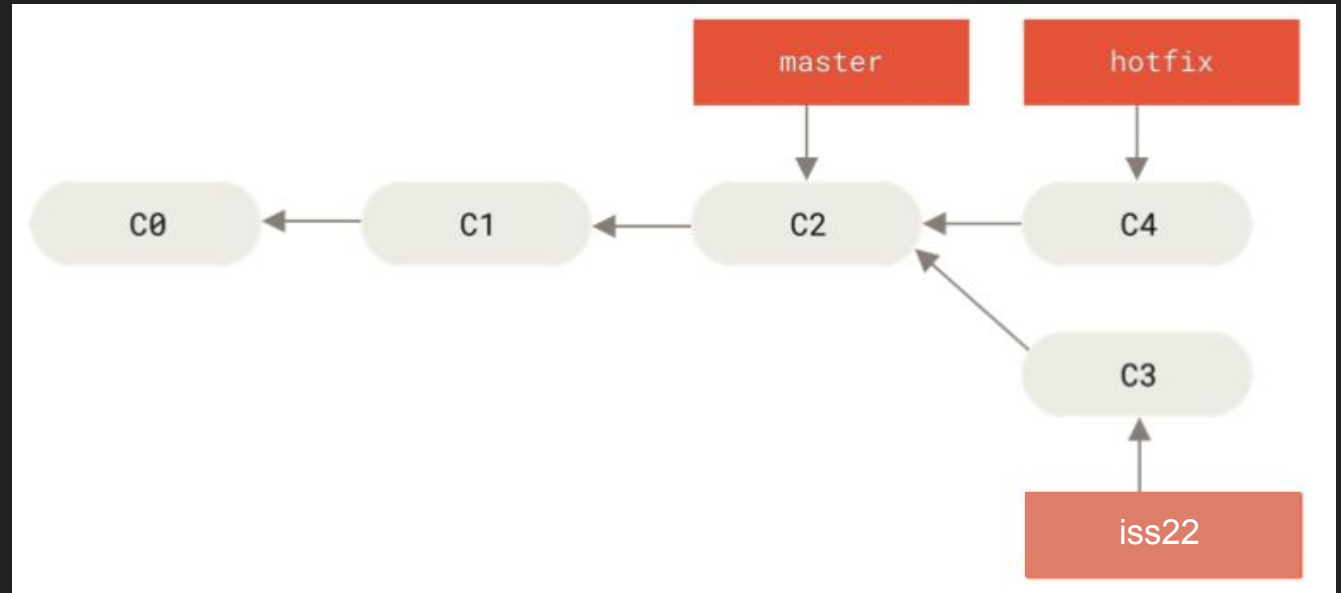
Branching and Merging

You can run your tests, make sure the hotfix is what you want, and finally merge the hotfix branch back into your master branch to deploy to production.

1. Run git checkout master
2. Run git merge hotfix

"Fast-forward" -   when you try to merge one commit with a commit that can be reached by following the first commit's history, Git simplifies things by moving the pointer forward because there is no divergent work to merge together

```
fabriziounicam:Local user$ git checkout master
Switched to branch 'master'
fabriziounicam:Local user$ git merge hotfix
Updating 2f45ef3..237308f
Fast-forward
 index.html | 1 +
 index.txt  | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 index.html
 create mode 100644 index.txt
```

Git

Up to now we have seen how to use git to do local versioning.

However we said during the last lesson that git is a distributed version control system and it is useful when collaborating with others.

How do you collaborate with others?   By means of the Internet

Remotes in Git

A remote repository is a repository stored somewhere else.

Most programmers use hosting services like:
- **GitHub**,
- **BitBucket,**
- **GitLab**

GitHub

The first thing you need to do is set up a free user account.

Visit https://github.com, choose a username that isn't already taken, provide an email address and a password, and click the big green "Sign up for GitHub" button.

Remote Repository

Create a new repository to share our code.

1. Click "New repository" button on the left side of the dashboard, or from the + button in the top toolbar next to your profile image as seen in the "New repository" dropdown.
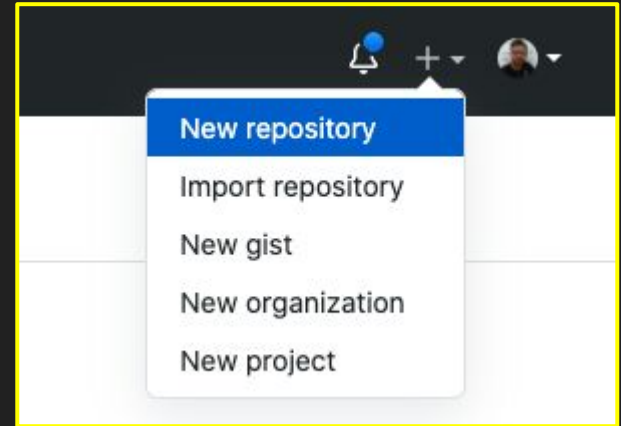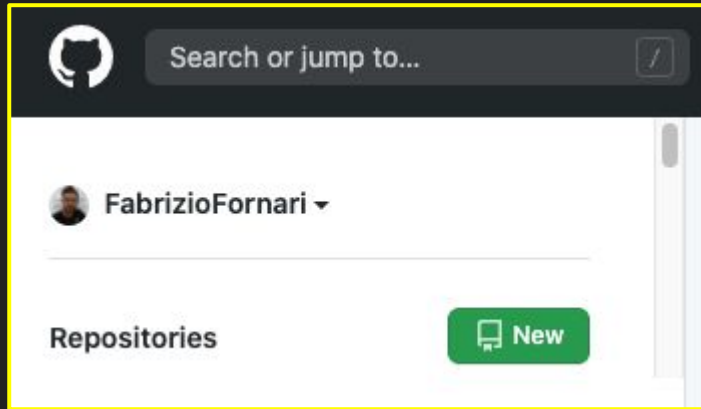
Remote Repository

1. Write the name of the repository
2. Add a description (optional)
3. Choose whether to create a Public or Private repository
4. Create the repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner *   **Repository name** *

👤 FabrizioFornari ▾   /
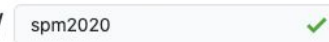
Great repository names are short and memorable. Need inspiration? How about **special-enigma**?

Description (optional)

⦿ 📖 **Public**
    Anyone on the internet can see this repository. You choose who can commit.

◯ 🔒 **Private**
    You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
    This is where you can write a long description for your project. Learn more.

☐ **Add .gitignore**
    Choose which files not to track from a list of templates. Learn more.
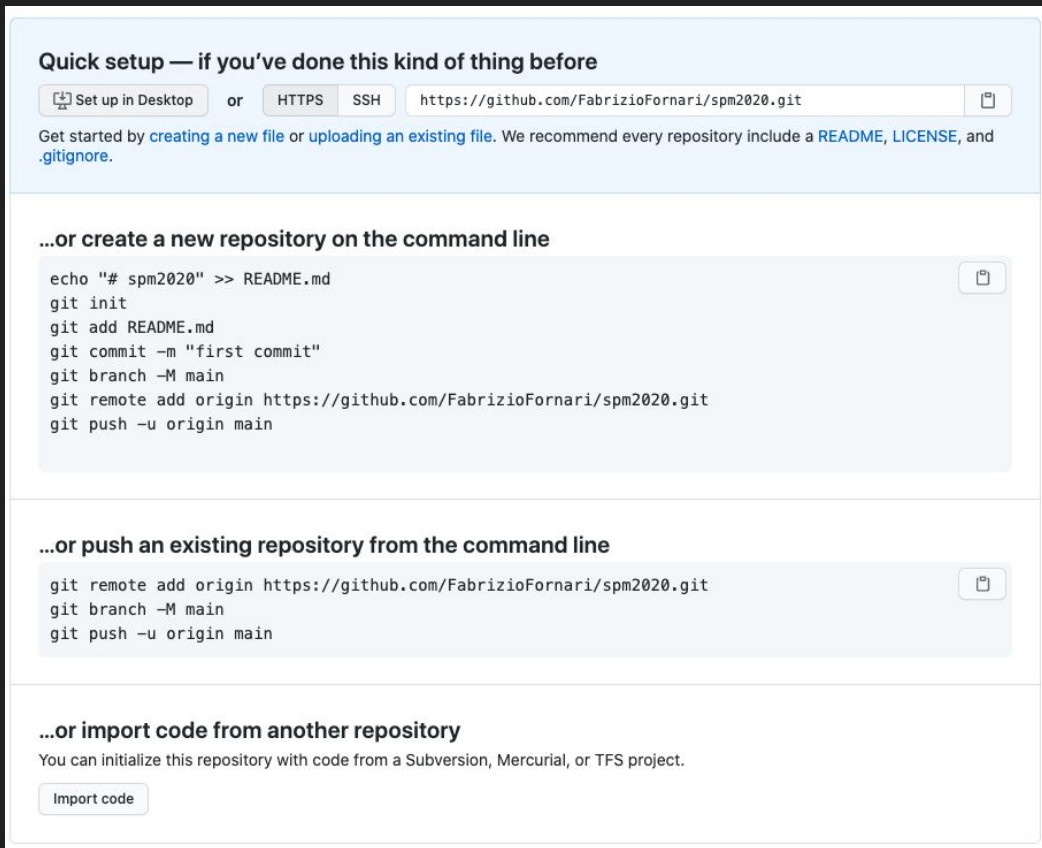
☐ **Choose a license**
    A license tells others what they can and can't do with your code. Learn more.
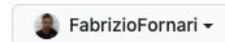
Create repository

Remote Repository

1. Write the name of the repository
2. Add a description (optional)
3. Choose whether to create a Public or Private repository
4. Create the repository

Remote Repository

As soon as the repository is created, GitHub displays a page with a URL and some information on how to configure your local repository.

Quick setup — if you've done this kind of thing before

Set up in Desktop or [HTTPS] [SSH] https://github.com/FabrizioFornari/spm2020.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# spm2020" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/FabrizioFornari/spm2020.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/FabrizioFornari/spm2020.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Remote Repository

What happened on the Server?

```
$ mkdir spm2020
$ cd spm2020
$ git init
```

But the repository now it is empty

Remote Repository

We have to link our local repository to the remote repository.

1. Access your local repository with a terminal (command line)
2. Run git remote add origin https://github.com/YourUserName/YourRepositoryName.git

```
git remote add origin https://github.com/FabrizioFornari/spm2020.git
```

We can check that the command has worked by running git remote -v

```
(base) fabriziounicam:MySecondRepo user$ git remote -v
origin  https://github.com/FabrizioFornari/spm2020.git (fetch)
origin  https://github.com/FabrizioFornari/spm2020.git (push)
```

The name origin is a local nickname for your remote repository. We could use something else if we wanted to, but origin is by far the most common.

Remote Repository

Once the nickname origin is set up, we push the changes from our local repository to the repository on GitHub

3. Run git push origin master

```
(base) fabriziounicam:MySecondRepo user$ git push origin master
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (15/15), 1.17 KiB | 399.00 KiB/s, done.
Total 15 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/FabrizioFornari/spm2020.git
 * [new branch]      master -> master
```

Remote Repository

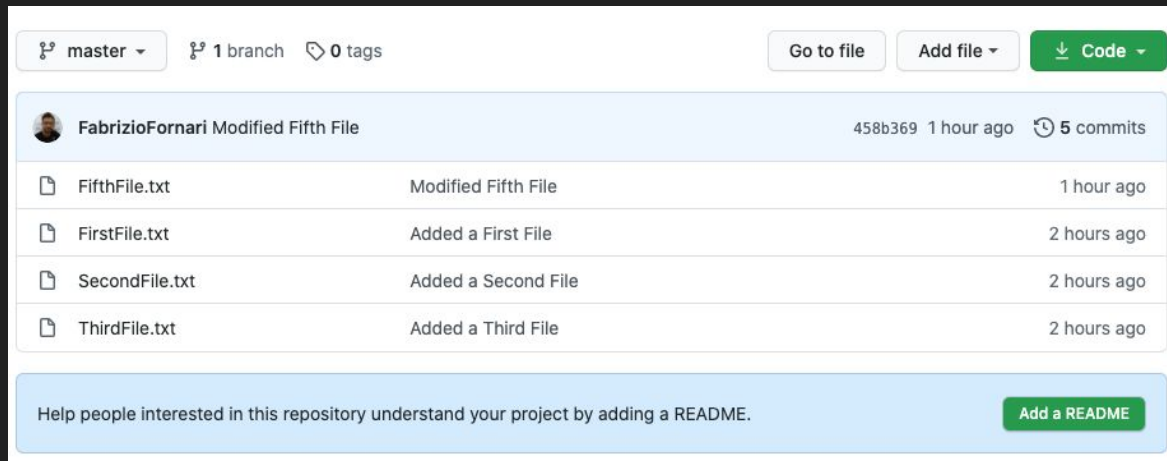Once the nickname origin is set up, we push the changes from our local repository to the repository on GitHub

3. Run git push origin master
4. Refresh your github webpage

Remote Repository

If you have more branches in your local repository and you want to keep track of all of them

5. Run git push origin --all

```
(base) fabriziounicam:MySecondRepo user$ git push origin --all
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 581 bytes | 290.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/FabrizioFornari/spm2020.git
 * [new branch]      hotfix -> hotfix
 * [new branch]      iss22 -> iss22
 * [new branch]      testing -> testing
```

Remote Repository

If you have more branches in your local repository and you want to keep track of all of them

5. Run git push origin --all

Commit vs Push

When we commit we update our local repository.

When we push we interact with the remote repository to update it with the changes we have made locally. In this way, who has access to the repository can see the changes.

How can your collaborators download the changes you have made and pushed to the remote repository?
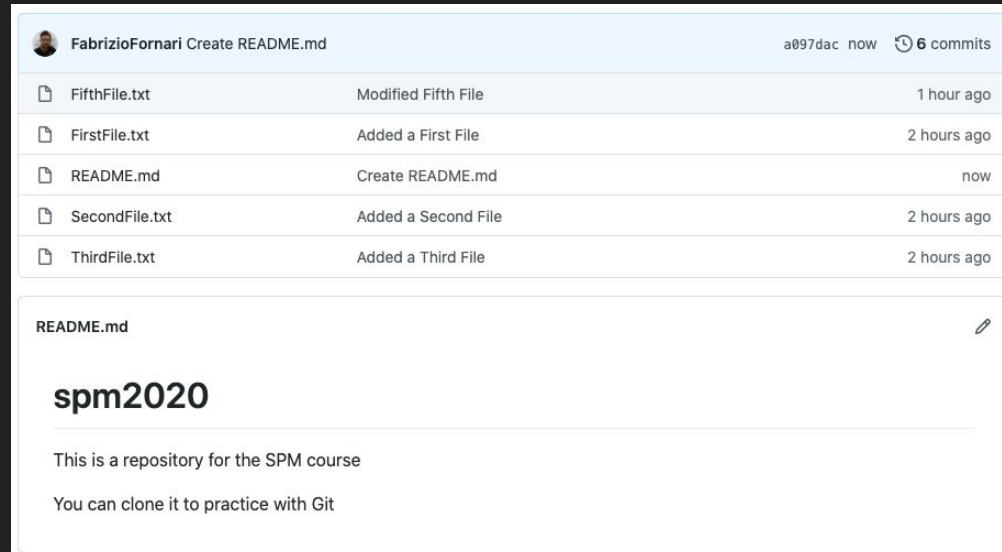
Git Pull

We can pull changes from the remote repository to the local one

1. Run git pull origin master

The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet.

I added a README file by means of the GitHub User Interface which allowed me to add the file and to perform a commit to store the changes

	FabrizioFornari Create README.md		a097dac now ⏱ 6 commits
📄	FifthFile.txt	Modified Fifth File	1 hour ago
📄	FirstFile.txt	Added a First File	2 hours ago
📄	README.md	Create README.md	now
📄	SecondFile.txt	Added a Second File	2 hours ago
📄	ThirdFile.txt	Added a Third File	2 hours ago

README.md

spm2020

This is a repository for the SPM course

You can clone it to practice with Git

Git Pull

We can pull changes from the remote repository to the local one

1. Run git pull origin master

The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet.
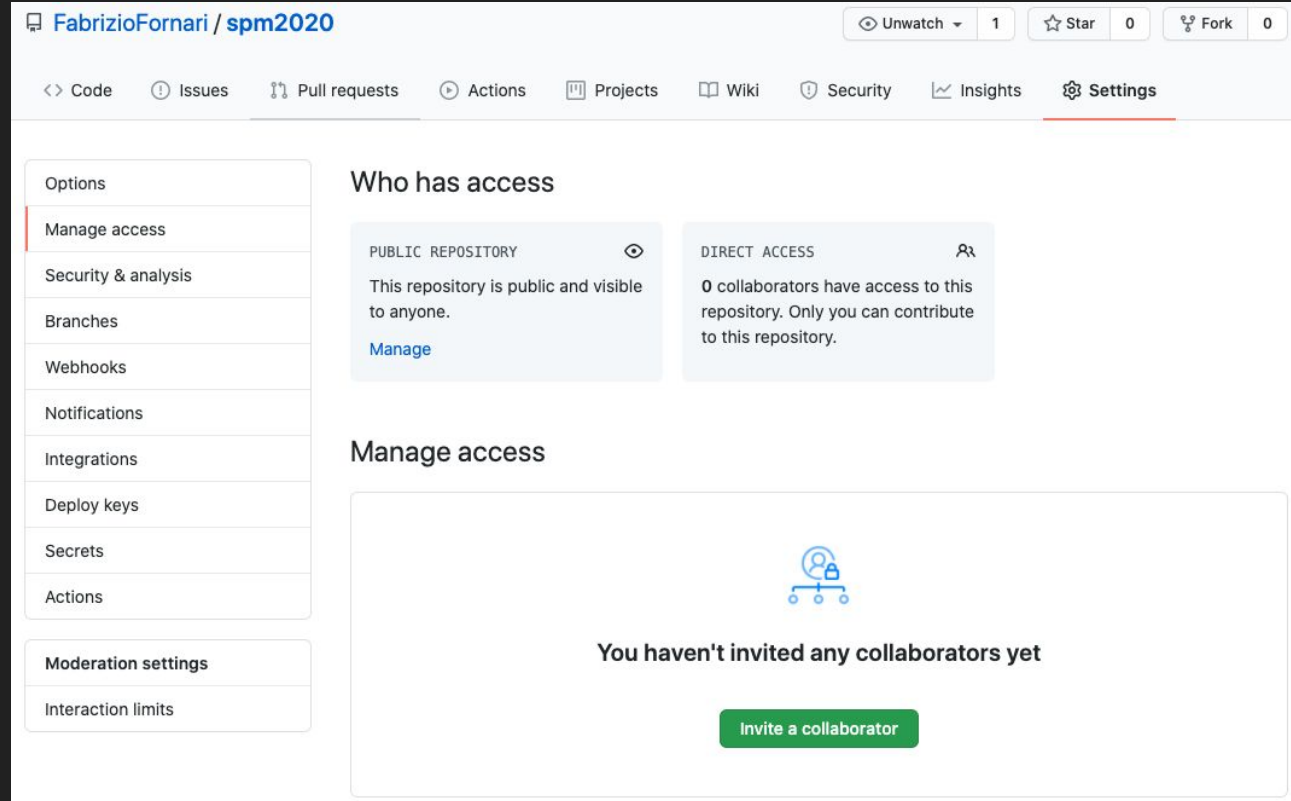
```
(base) fabriziounicam:MySecondRepo user$ git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/FabrizioFornari/spm2020
 * branch            master     -> FETCH_HEAD
   7cc6357..965fc6e  master     -> origin/master
Updating 7cc6357..965fc6e
Fast-forward
 README.md | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 README.md
```

Collaboration on GitHub

One person will be the "Owner" and the others will be the "Collaborators".

The Owner needs to give the Collaborators access.

The Collaborators will receive an email and/or they can check notifications on https://github.com/notifications

Collaboration on GitHub

A collaborators can download a copy of the Owner's repository to their machine. This action is referred as "cloning a repository".

To clone a Git project hosted on GitHub:

1. Run git clone <address of the project>.git <pathToAFolderInYourMachine>

    You can run git clone https://github.com/FabrizioFornari/spm2020.git

```
(base) fabriziounicam:SPM user$ git clone https://github.com/FabrizioFornari/spm2020.git GitHubRepo/
Cloning into 'GitHubRepo'...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 31 (delta 11), reused 18 (delta 4), pack-reused 0
Unpacking objects: 100% (31/31), done.
```

Collaboration on GitHub

The collaborator can now make a change to the copy of the repository he/she has.


```
(base) fabriziounicam:GitHubRepo user$ git commit -m "Added a Eight File"
[master fe23e6d] Added a Eight File
 1 file changed, 1 insertion(+)
 create mode 100644 EigthFile.txt
(base) fabriziounicam:GitHubRepo user$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 297.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/FabrizioFornari/spm2020.git
   965fc6e..fe23e6d  master -> master
```

Note: here I am using the same account but the effect would be the same by using the account of a collaborator

Collaboration on GitHub

To update the Owner's version, the Owner runs:

1. git pull origin master

```
[(base) fabriziounicam:MySecondRepo user$ git pull origin master
From https://github.com/FabrizioFornari/spm2020
 * branch            master      -> FETCH_HEAD
Updating 965fc6e..fe23e6d
Fast-forward
 EigthFile.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 EigthFile.txt
```

Collaborative Workflow

It is good to be sure that you have an updated version of the repository you are collaborating on, so you should git pull before making our changes.

The basic collaborative workflow would be the following.

- Run git pull origin master to update your local repo with changes stored on the server
- Make changes and stage them by running git add
- Commit your changes by running git commit -m
- Upload the changes to GitHub by running git push origin master

Better to make many commits with small changes rather than of one single commit with massive changes: *small commits are easier to read and to review*. Pay attention to this for the final evaluation.

Git Fetch

On the command line, the Collaborator can run git fetch origin master to get the remote changes into the local repository, but without merging them.

Then by running git diff master origin/master the Collaborator will see the changes output in the terminal.

$ git fetch origin/master
$ git diff master origin/master
$ git merge origin/master

The Dark Side of Collaboration

When you are collaborating on a shared repository you may end up changing the same files other collaborators are working on. Version control helps us to manage these conflicts by giving us tools to resolve overlapping changes.

Note: this can also happen if you are the only one working on a repo but, for instance, on different devices (a laptop and a pc)

The Dark Side of Collaboration

As collaborators try to modify the same file (e.g. add a different line of text/code).

Collaborator A

```
(base) fabriziounicam:MySecondRepo user$ vi EigthFile.txt
(base) fabriziounicam:MySecondRepo user$ git add EigthFile.txt
(base) fabriziounicam:MySecondRepo user$ git commit -m "as collaborator A I modified file Eight"
[master 49a3087] as collaborator A I modified file Eight
 1 file changed, 1 insertion(+)
(base) fabriziounicam:MySecondRepo user$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 335 bytes | 335.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/FabrizioFornari/spm2020.git
   fe23e6d..49a3087  master -> master
```

The Dark Side of Collaboration

As collaborators try to modify the same file (e.g. add a different line of text/code).

Collaborator B

```
(base) fabriziounicam:GitHubRepo user$ git mv EigthFile.txt EightFile.txt
(base) fabriziounicam:GitHubRepo user$ git add .
(base) fabriziounicam:GitHubRepo user$ git commit -m "as collaborator B I fixed a typo to the name of file eight"
[master c05d6f2] as collaborator B I fixed a typo to the name of file eight
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 EightFile.txt
 delete mode 100644 EigthFile.txt
(base) fabriziounicam:GitHubRepo user$ git push origin master
To https://github.com/FabrizioFornari/spm2020.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/FabrizioFornari/spm2020.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Git rejects the push because it detects that the remote repository has new updates that have not been incorporated into the local branch.

Solve the Conflict

What we have to do is pull the changes from GitHub, merge them into the copy we're currently working in, and then push that.

```
(base) fabriziounicam:GitHubRepo user$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/FabrizioFornari/spm2020
 * branch            master      -> FETCH_HEAD
   fe23e6d..49a3087  master      -> origin/master
CONFLICT (modify/delete): EigthFile.txt deleted in HEAD and modified in 49a3087ebf5f079c13fa28427a83b4a353ed1043.
Version 49a3087ebf5f079c13fa28427a83b4a353ed1043 of EigthFile.txt left in tree.
Automatic merge failed; fix conflicts and then commit the result.
```

Solve the Conflict

Run git status


```
(base) fabriziounicam:GitHubRepo user$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add/rm <file>..." as appropriate to mark resolution)

        deleted by us:   EigthFile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Anything that has merge conflicts and hasn't been resolved is listed as unmerged. Git adds standard conflict-resolution markers to the files that have conflicts, so you can open them manually and resolve those conflicts.

Solve the Conflict

I decided to merge the content of the files by copy paste. I did so by using my text editor
open -a TextEdit File.txt

I Run git status

```
(base) fabriziounicam:GitHubRepo user$ ls
EigthFile.txt    EigthFile.txt    FifthFile.txt    FirstFile.txt    README.md        SecondFile.txt    ThirdFile.txt
(base) fabriziounicam:GitHubRepo user$ open -a TextEdit EightFile.txt
(base) fabriziounicam:GitHubRepo user$ open -a TextEdit EigthFile.txt
```

```
(base) fabriziounicam:GitHubRepo user$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add/rm <file>..." as appropriate to mark resolution)

        deleted by us:   EigthFile.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   EightFile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Solve the Conflict

I removed EigthFile.txt

I run git status

I added the changes
to EightFile.txt

I committed the
changes

I pushed the changes

```
(base) fabriziounicam:GitHubRepo user$ git rm EigthFile.txt
EigthFile.txt: needs merge
rm 'EigthFile.txt'
(base) fabriziounicam:GitHubRepo user$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   EightFile.txt

(base) fabriziounicam:GitHubRepo user$ git add EightFile.txt
(base) fabriziounicam:GitHubRepo user$ git commit -m "merged EightFile with EigthFile which has been removed"
[master 3697611] merged EightFile with EigthFile which has been removed
(base) fabriziounicam:GitHubRepo user$ git push origin master
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 588 bytes | 588.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/FabrizioFornari/spm2020.git
   d2d1482..3697611  master -> master
```

Solve the Conflict

Good Practice

- Pull from upstream more frequently, especially before starting new work
- Use topic branches to segregate work, merging to master when complete
- Make small commits
- Break large files into smaller ones so to reduce the possibility of conflicts

About Conflicts:
- Clarify who is responsible for what areas with your collaborators
- Discuss the order of tasks with your collaborators so that tasks expected to change the same lines won't be worked on simultaneously
- If the conflicts are stylistic churn (e.g. tabs vs. spaces), establish a project convention that is governing and use code style tools (e.g. htmltidy, perltidy, rubocop, etc.) to enforce, if necessary

Remember the Workflow
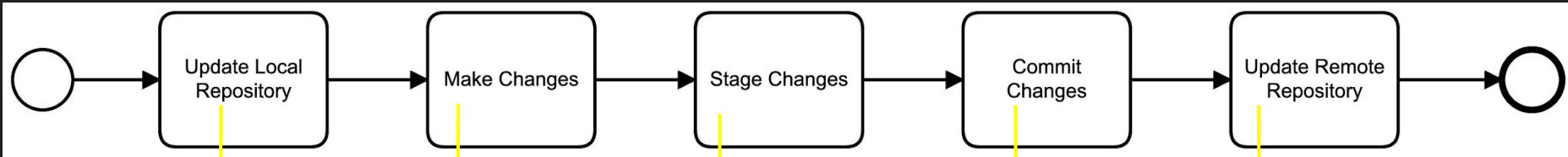
Additional Materials

Pro Git
https://git-scm.com/book/en/v2
by Scott Chacon and Ben Straub

In case of...