Introdução

Lecture 1 — 21/10/2021

Course Content

- Git / Java / Gradle
- Unit Testing / Test Driven Development
- SOLID Principles
- UML: Class and Sequence Diagrams
- Design Patterns
- Refactoring and Code Smells

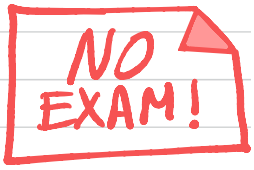
Main Bibliography

- Bruce Eckel; ThinKing in Java
- Russ Miles and Kim Hamilton; Learning UML 2.0.
- Kent Beck; Test-driven development
- Eric Gamma... [et al.]; Design Patterns.
- Martin Fowler; Refactoring

Evaluation

- You must obtain a minimum of 40% in all evaluation components.

(10%)	Project (60%)			(30%)
Class participation	Intemediate Report (10%)	Final Report (30%)	Code (60%)	Multiple Choice Quiz

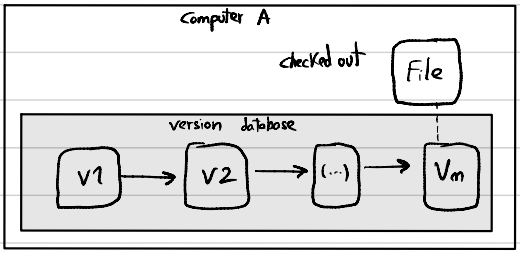


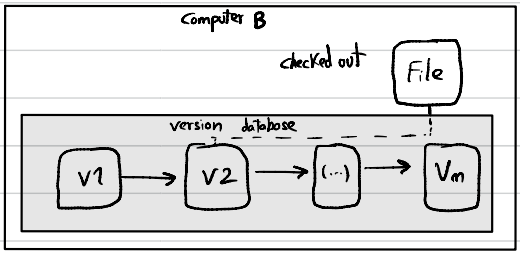
Communication

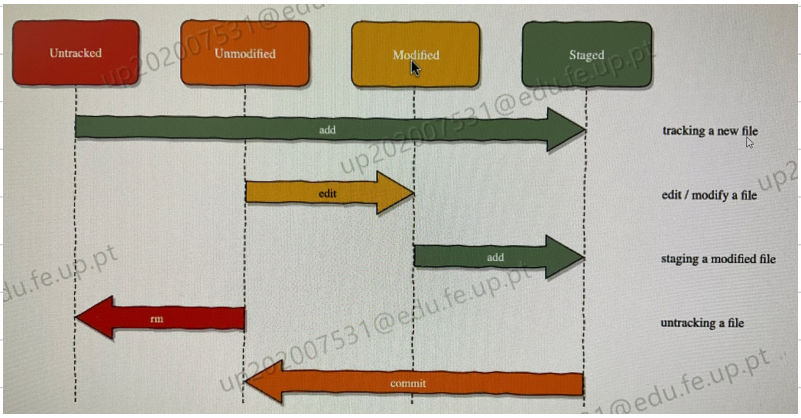
SLACK

Git

Version Control Systems (VCS)







Create a Repository

Enter a local directory, currently not under version control:

```
cd project
```

And turn it into a Git repository:

```
git init
```

This will create an hidden *.git* subdirectory containing all of your necessary repository files.

Add

The add command can be used to:

1. Track and stage a file that is currently not tracked by Git.
2. Stage a file that has been modified.

```
$ echo "hello git" > README    # File is created
$ git add README               # File is now tracked and staged
```

You can use the --all or -A flag to stage all untracked or modified files.

```
$ echo "hello git" > README    # File is created
$ git add --all                # File is now tracked and staged
```

Commit

The commit command records a new snapshot to the repository:

```
$ echo "hello git" > README    # File is created
$ git add README               # File is now tracked and staged
$ git commit                   # Commits the file
```

After running commit, Git will open your predefined text editor so that you can write a small commit message (or use the --message or -m flag).

The --all or -a flag automatically stages any modified (tracked) files:

```
$ echo "goodbye git" > README      # Already tracked file is modified
$ git commit -a -m "Edited README" # Stages and commits the file
```

Status

The status command can be used to determine which files are in which state:

```
$ echo "hello git" > README    # File is created
$ git status                   # Asking for file status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README

nothing added to commit but untracked files present (use "git add" to track)
```

The --short (or -s) flag can be used to get a more concise output:

```
$ git status --short           # Asking for file status
?? README
$ git add README               # File is now tracked and staged
$ git status --short           # Asking for file status
A  README
```

Partially Staged Files

A file can be partially staged:

```
$ echo "some text" > README      # File is modified
$ git add README                 # Modifications are staged
$ echo "another text" >> README  # File is modified again
$ git status -s
AM README                        # Added to staging area and modified
```

1) Commiting again would only commit the initial staged edits:

```
$ git commit -m "Added some text"     # Commiting initial edit
$ git status -s
 M README
$ git add README                      # File now still has changes
M  README                             # Staging those changes
$ git commit -m "Added another text"  # Commiting following edits
```

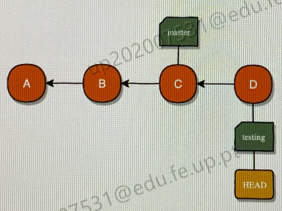
2) We could also add the new modifications first and only commit once:

```
$ git add README                          # Staging following changes
$ git status -s
A  README                                 # All changes staged
$ git commit -m "Added some and another text"  # Commiting both changes at once
```

Moving the HEAD

If we create a new commit now:

```
$ echo "more license info" >> LICENSE
$ git commit -a -m "Testing LICENSE"
```



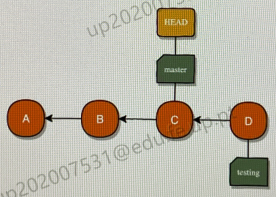
We can see that only the current branch, the one pointed by the HEAD, moved.

Checkout

If we checkout the *master* branch again, two things happen:

```
$ git checkout master
```

1. The HEAD moves to the commit pointed by the master branch.
2. Our files are reverted to the snapshot that master points to.

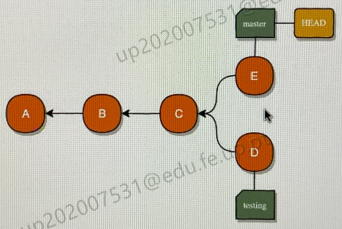


This means we are now working on top of a version that has already been changed. Any changes we make will create a divergent history.

Divergent Histories

Now that we are back to our master branch, let's do some more changes:

```
$ git checkout master
$ echo "license looks better this way" >> LICENSE
git commit -a -m "Better LICENSE"
```



Now we have two divergent histories that have to be merged together.

Merging

Merging is done by using the merge command:

```
$ git checkout master
$ git merge testing
```

Git merges the identified branch into the current branch.

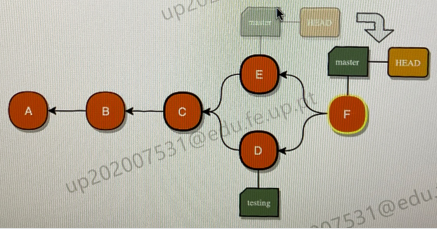
Git uses two main strategies to merge branches:

- Fast-forward merge: when there is no divergent work
- Three-way merge: when there is divergent work

Three-way Merge

When the commit on the branch you're on isn't a direct ancestor of the branch you're merging in, Git uses the two snapshots pointed to by the branch tips and the common ancestor of the two to create a new commit.

```
$ git checkout master
$ git merge testing
```



Deleting Branches

If you do not need a branch any longer, you can just **delete it.**

Deleting a branch leaves all commits alone and only **deletes the pointer.**

```
$ git branch -d testing
```

Conflicts

If you changed the **same part** of the **same file** differently in the two branches you're merging, Git **won't be able** to merge them cleanly:

```
$ git merge testing
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```

You can use the **status** command to see which files have conflicts:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

        both modified:   README

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

Resolving Conflicts

Editing the file with conflicts we can see the conflict:

```
This is a README file
<<<<<<< HEAD
This was added in the master branch
=======
This was added in the testing branch
>>>>>>> testing
```

To solve it we just have to edit the file:

```
This is a README file
This was added in the master branch
This was added in the testing branch
```

And commit the merge:

```
$ git commit
```

Git Ignore

- A *.gitignore* file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected.
- Each line in a *.gitignore* file specifies a pattern.
- Some examples:

```
# this is a comment
docs/          # everything inside root directory docs
**/docs/       # any docs directory
!docs/**/*.txt # don't ignore (!) any .txt files inside directory docs
```

What files to ignore: 1) not used by your project, 2) not used by anyone else and 3) generated by another process.

Remots

Remotes

Remote repositories are versions of your project that are hosted elsewhere (another folder, the local network, the internet, ...).

You can push and pull data to and from remotes but first you need to learn how to configure them properly.

Protocols

Git can use four major network protocols to transfer data to and from remotes:

- Local - Useful if you have access to a shared mounted directory.
- Git - A special daemon that comes packaged with Git. SSH but without authentication or encryption.
- SSH - The most commonly used protocol.
- HTTP - Easiest to setup for read-only scenarios but very slow.

Adding Remotes

Besides the origin remote from where we cloned our project, we can add more remotes:

```
git remote add john http://john-laptop.org/test-repository
```

In this example, we added a new remote and gave it the alias *john*:

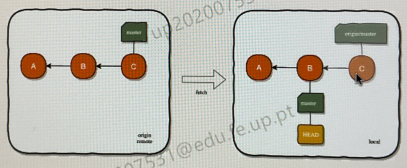
```
$ git remote -v
origin  https://example.com/test-repository (fetch)
origin  https://example.com/test-repository (push)
john  http://john-laptop.org/test-repository (fetch)
john  http://john-laptop.org/test-repository (push)
```

Fetching

Fetching gets all the data from a remote project that you don't have yet.

After fetching, you will also have references to all the branches from that remote.

```
$ git fetch origin
```

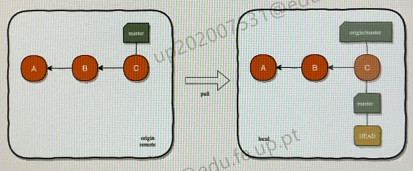


Fetching only downloads the data to your local repository. It doesn't automatically merge it with any of your work or modify what you're currently working on.

Pulling

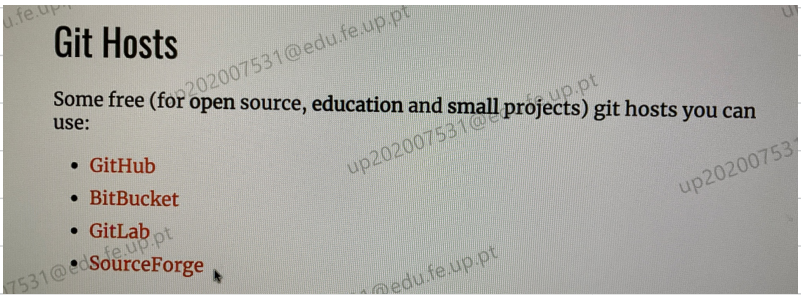
If your current branch is set up to track a remote branch, you can use the git pull command to automatically fetch and then merge that remote branch into your current branch.

```
$ git pull origin master # fetches and merges origin/master
```



This fetches data from the server you originally cloned from and automatically tries to merge it into the code you are currently working on.

```
$ git pull # uses default values for current local branch
```


Git Hosts
Some free (for open source, education and small projects) git hosts you can use:
GitHub
BitBucket
GitLab
SourceForge