TP Git

1 Foreword

The objective of this practical class is to manipulate *git* in a real life manner with a dummy C++ project hosted on **github**. We will use both the command line and a graphical user interface for windows: **GitExtensions**. The reader is supposed to have basic knowledge of git concepts (see for instance the ProGit book by Scott Chacon or the brilliant lesson by Matthieu Durut at ENSAE).

The project will have two committers so this practical is supposed to be followed by a pair of student. The first student of the pair will be referred to as `StudentA` in the following while the second one will be `StudentB`.

2 Student A

2.1 Fork an existing repository and invite a collaborator

- Open your web browser and visit github.com. Create a free **github** account and stay logged.
- Go to url *github.com/bpatra/tpgit* and fork the repository.
- Make sure that `StudentB` has completed 3.1. Then in your newly forked *tpgit* repository go to *settings, collaborators* and add `StudentB`.

2.2 Clone the repository locally with command line

- On windows start button look for the *Command Prompt* and run it.
- Change current directory to the location where you would want to put the local repository. You will not have to create a folder it will be created automatically (see below).
- Clone the remote repository by executing the following command:
  git clone https://< studenta>@github.com/studenta/tpgit.git
  Until the end of this this document, all command line instructions should be executed with the root of the local repository as command line current directory.

2.3 First commit with the command line

- On *ArithmeticMethods.h* comment out the functions.
  e.g. *//IsPrime checks if a positive integer is a prime number. Returns true if the input is prime and false if not.*
- View the diff of what you have modified so far by running *git diff*.
- Check the status of your repository by running *git status*.
- Add the non staged changes of *ArithmeticMethods.h* in the staging area by executing
  git add ./TP1/ArithmeticMethods.h
  Note: we may use alternatively *git add -i*.
- Commit your work by running *git commit -m "commenting ArithmeticMethods header file"*

2.4 Second commit with GitExtensions

Follow instructions from 3.3 but comment the functions declarations of the *B.h* file instead.

2.5 Pushing your modification to the remote

- On **GitExtensions** open *Commands* > *Push*.
- Enter your credentials.
- It works simply for the first time (because the remote branch did not have any new commits so the push is a fast forward merge). Next time you will probably have to fetch or pull to be able to push to **github** repository. Check 3.5.

2.6 Basic local branching, reintegrate by merging

- On **GitExtensions** right click on commit `aaba7414952b` by benoit and create a new branch named "vehicule"
- Create a class called Vehicule
- Create a commit with an appropriate message.
- Create a child class of the class Vehicule called Car.
- Create a commit.
- With **GitExtensions** merge your current branch ("vehicule") with the branch "master", use *Commands* > *Merge*.
- Push your changes to **github**.

2.7 Basic local branching, reintegrate by rebasing

- On **GitExtensions** right click on commit `aaba7414952b` by benoit and create a new branch named "complex"
- Create a class Complex representing a complex number add constructor and basic accessors.
- Create a commit. Mark the commit Hash (SHA1) somewhere.
- Extend the Complex class so that a new method getModulus returns the modulus of the current instance.
- Create a commit.
- With **GitExtensions** rebase your current branch ("complex") with the branch "master", use *Commands* > *Rebase*.
- Compare the commits hash that you have noted above and the hash of the created commits after the rebase. Compare the rebase operation with merge.
- Push your changes to **github**

2.8 Solving a conflict

Synchronize with `StudentB` and wait him/her to finish 3.7. You and `StudentB` will perform actions on the same file. You will push the changes before him so the conflict will be for him to solve.

- In the function definitions in PointerManipulation.cpp change the name of the variable from n to p.
- Create a commit and Push it to **github**.

2.9 Viewing who has done what: blaming

- On the **GitExtensions** main window visit the tab *File tree*.
- Find in the tree view ArithmeticsMethods.h (or any file of your choice).
- Right click *Blame*.

2.10 Rewriting local history: rebasing interactively

For this part we will suppose that we want to change our convention about the include guards. The new convention for a file named A.h the preprocessor constant will be named A_h rather than A_H.

- Apply the new conventions to all files in the solution. But leave one intentionally, as if you have forgotten it.
- Create a commit with a message "Apply new include guards conventions"
- Change something else in the project. For example change the message in B.cpp. Create a commit.
- Apply the include guards conventions for the forgotten file. Create a commit with message "fixup forgotten file for include guard convention".

This is something quite frequent, we have forgotten something and our history of commits is not the one we wanted. What we would like to do is to rearrange the commit so that the third commit is “integrated” with the first (squash). Even if it does not seem possible, it is, as long as we have not shared any of those commits. To this aim we will run the command: *git rebase -i* command.

- Execute in command line git rebase -i HEAD˜ 3. You will have a text editor popping. Read carefully the comments in the file.
- Cut the line referring to the forgotten commit and paste it right under the line of first commit. (Do not touch anything else !)
- Change also for this line the instruction *pick* -> *fixup*.
- Save file and close the window, let git do the job.

You may also try without the command line with the Rebase command of **GitExtensions**. You will have to check the options "interactive" and "specific ranges" (pickup the commit just before the one with the include guards).

3 Student B

3.1 Create a github account

1. Open your web browser and visit github.com. Create a free **github** account and stay logged.

3.2 Clone the repository with command line

Same as 2.2 but with the command line git clone https://**<studentb>**@github.com/studenta/tpgit.git

3.3 First commit with GitExtensions

- On *PointersManipulation.h* comment out the functions e.g. for the third one:
  //Pass the argument of the function by reference.
- Open **GitExtensions** and then choose to open the local repository.
- Click commit, view the diff, stage the modified file and commit with an appropriate message.

3.4 Second commit with command line

Follow instructions from 2.5 but comment something on the file *A.h*.

3.5 Pushing your modification to the remote

Make sure `StudentA` has completed 2.5

- On **GitExtensions** open *Commands* > *Push*
- Enter your credentials
- If `StudentA` has completed 2.5 then git refuses you push because you cannot fast forward merge on branch origin/master. Solve this by executing the following command line
  git fetch origin/master then *git merge origin/master* or simply by using Pull in **GitExtensions**.
- Push again.

3.6 Basic local branching, reintegrate by rebasing

- On **GitExtensions** right click on commit `aaba7414952b` by benoit and create a new branch named "person"
- Create a class Person representing the an individual that has two members Name and Age.
- Create a commit. Mark the commit Hash (SHA1) somewhere.
- Extend the Person with a new method *Hello*, which returns a basic string such as "Hello I am <the name here> and I am <the age here>".
- Create a commit.
- With **GitExtensions** rebase your current branch ("person") with the branch "master", use *Commands* > *Rebase*.
- Compare the commits hash that you have noted above and the hash of the created commits after the rebase.
- Push your changes to **github**

3.7 Basic local branching, reintegrate by merge

- On **GitExtensions** right click on commit `aaba7414952b` by benoit and create a new branch named "boats"
- Create a class called Boat
- Create a commit.
- Create a child class of the class *Boat* called *MotorBoat*.
- Create a commit.
- With **GitExtensions** merge your current branch ("vehicule") with the branch "master", use *Commands* > *Merge*.
- Push your changes to **github**.

3.8 Solving a conflict

Synchronize with `StudentA` and wait him/her to finish 2.7.

- In the function definitions in PointerManipulation.cpp change the name of the variable from n to m.
- Create a commit.
- Wait that `StudentA` has performed 2.8.
- Execute the Pull command.
- You should have a conflict. Indeed your work conflicts with the commit from `StudentA`! **GitExtensions** will help you to solve it (click on unresolved conflicts). You will have to choose between your version with m or the one of `StudentA` with p. Suppose that your are right and fix the conflict and always use the "m version".

3.9 Viewing who has done what: blaming

Same as 2.9.

3.10 Rewriting local history: rebasing interactively

Same as 2.10, do not push commits once done.