Object Oriented Programming with Python Lecture 2

Rogelio A Mancisidor
Department of Data Science and Analytics
BI Norwegian Business School

31 August 2022

Today

Outline

Cost of owning a car

Problem 10, chapter 2: Write a program with inputs

- Cost
- Km per year
- Efficiency in km./L
- Gasoline price

Calculate the cost of owning a car for 5 years

Outline

Is this familiar?

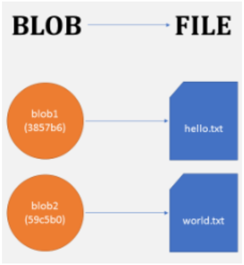
```
my_file.py
my_filev2.py
my_filev3.py
my_file_final.py
my_file_final_final.py
```

Git terminology

- Git tracks the history of a collection of files and folders
- A file is called *blob*, which is just a bunch of bytes
- A directory is called *tree*, it maps names to blobs
- A *snapshot* is the top-level tree that is being tracked
- Snapshots in Git refer to a set of *parents*
- Git calls these snapshots *commits*

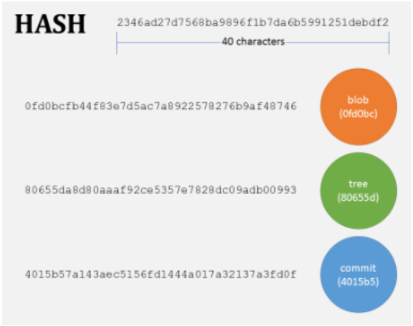
Blob

- When files are changed, a new blob store the new file with all changes



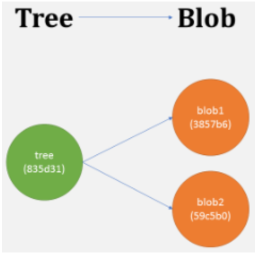
Hash

- All Git objects (blob,tree,commit) are identified by SHA-1 hash


HASH
2346ad27d7568ba9896f1b7da6b5991251debdf2
40 characters
0fd0bcfb44f83e7d5ac7a8922578276b9af48746
blob (0fd0bc)
80655da8d80aaaf92ce5357e7828dc09adb00993
tree (80655d)
4015b57a143aec5156fd1444a017a32137a3fd0f
commit (4015b5)

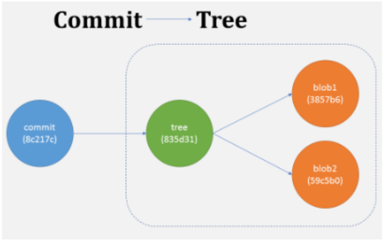
Tree

- Files are stored in a directory or folder
- Similarly, a tree in Git represents directories for blobs and more trees



Commit

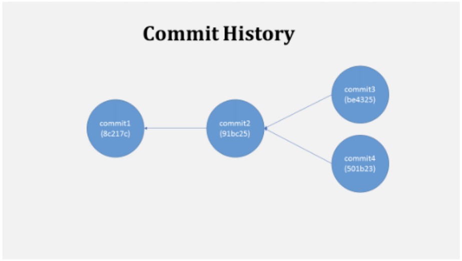
- Suppose we have 2 files. We want to save them in a way that we can recover an exact snapshot of them
- Each commit points to the tree of 2 blobs, with its own SHA-1 hash



- If we change blob1, Git stores a new blob (blob1_b) and a new tree is created with a new hash, pointing to blob1_b and blob2.

History

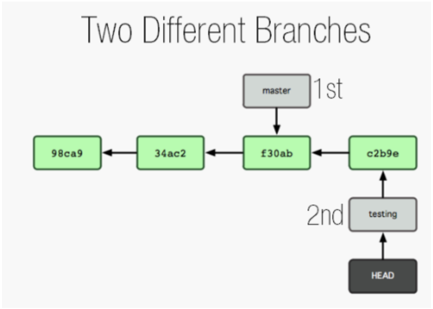
- As we make changes, new commits are created pointing to the previous commit
- A chain of commits maintains the history of what was done



- We can always load any previous commit

Branch

- A branch is a move-able pointer, pointing to the latest commit in that branch
- Head always points to the branch in which we are
- You can work on each branch independently



Outline

Github

- Create a `github` account with the user name `SXXXXXXX` (your student id)
- Choose the free subscription
- In that account, create a repository called `GRA4152` (see next slide)
- Create a token to be able to push commits, see here
- Save your token! you will need it to push commits. Optionally, setup a SSH identification protocol.
- Make sure you have installed `git`, otherwise see here

Setting up git and GitHub

New repo from an exsisting project

1. Create a directory called `GRA4152` and `cd` into it
2. Type `git init`
3. Type `git add` to add all of the relevant files.
4. Type `git config --local user.name "SXXXXXXX"`
5. Type `git config --local user.email "yourmail@bi.no"`
6. Type `git commit -m "initial commit"`

Push to `GitHub`

1. Go to your account, click `+`, `new repository`, and `create repository`
2. Type `git remote add origin git@github.com:<user>/<name>.git`
3. Type `git push -u origin master`
4. Use `SXXXXXXX` and the token generated to identified yourself

Useful commands

Some useful git commands

```
git clone https://github.com/<user>/<name>.git # Clone repository locally
git status           # Show the working tree status
git diff             # Show changes not yet staged
git add <file>       # Add <file> to stage
git log              # Show commit logs
git restore <file>   # Undo changes not yet staged
```

If you are interested in git, see Eficode Academy

Basic use

For tracking changes in your code and make them available in GitHub

```
git pull  # only if you work on different pc's
git add . # stage all modified files
git commit -m "add an explanation"
git push  # only to publish changes in GitHub
```

Stashing

Stashing temporarily shelves changes so you can work on something else. It is specially useful if you need to work on something else and are not (or do not want) ready to commit. Specially useful in OOP. Note, be careful with it!

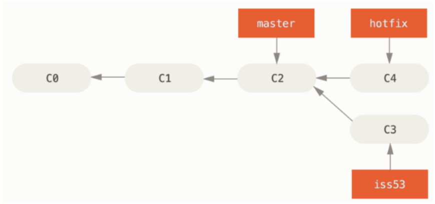
```
git stash            # save uncommitted changes
git start list       # list all stashes
git stash pop        # reapply changes to your working copy
git stash drop       # delete stash
```

Branching

Branching is probably the cleanest way to test if a new approach works, without messing up with a running code.

```
git checkout -b <branch_name> <from_branch>
# add files or modified current files
git commit -m "changes in new branch"
git checkout master               # switch to master branch
git merge <branch_name>           # fast-forward merge
git branch -d <branch_name>       # delete branch
```

If you make any modification in the new branch, it must be committed before switching into another branch!



Pull request

Suggest changes on somebody else repo. First, fork a repo in GitHub (upper right corner)

```
git clone https://github.com/<your_user>/<name>.git # Clone repository locally
git checkout -b <branch_name> <from_branch>
git remote add upstream https://github.com/<orig_user>/<orig_repo>
# Modify(add) some(new) files
git status # should display the file(s) modified
git add .  # stage all modified files
git commit -m "added some suggestions"
git push -u origin <branch_name>
```

Now you will see a "Compare & pull request" green button in GitHub. Click it and open a pull request by clicking "Create pull request"