Notes on GIT

Author: Marco

Some notes on git while listening to the lecture Version Control (git). There is also a good tutorial by the same guy I think. I stole some stuff from there!

Start Intro

- git is the *de-facto standard* for version control
- git is rather complicated and it's maybe not the best to learn git top-down (just learn some commands), but bottom-up, learn git and the data model and then the commands
- what is version control?
  - worst version: Manual version control: Save file, send to collegues etc
  - a bit better: stuff like dropbox (DON'T USE GIT WITH DROPBOX)
  - best: overleaf (since can work simultaniously), but it's on a special server etc and does not work for mathematica stuff. Also cannot really try to implement a new feature and maybe just throw it away if you don't like it or give up.
  - git
- what about git and github?
  - git is the underlying stuff and github is a server that you sinc your git stuff to
  - github is a repository host for git
  - git to github is like porn to pornhub
  - git is independent of github. You can use it without any internet and there are also other hosters like gitlab or it's also not too hard to host it yourself on some server.
- you track some specific folder (and its subfolders and files) with git
- Draw

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

- Explanation
  - "tree" = folder
  - "blob" = file
  - not the whole folder is saved every time you save, but it just tracks the changes of the files!
  - Each snapshot has a parent and changes
  - each blob/tree/commit has a *hash* (`sha1`, 40chars)

- – "Objects are addressed by their hash"
- Modelling the history
  - – Draw the diagram of the history First Example ———————————
    —————-
- `git init`
- `git status`
- `touch hello.txt`
- `git add hello.txt`
- `git commit`
- show hashes with git log
- can show hashes with `git cat-file -p hash`. E.g. go to a commit, then the tree and there the blob.
- right now I am only showing git, not github

References

- references: map from string to string: `fixEncodingBug -> hash`

  - – human readable strings to refer to the hashes

- `master` is a pointer ("reference") to the commit with some hash

- `HEAD` is where you are currently looking right now

-

Continue first example Example

Just go on with the first example. Draw the tree while going through this example

- `ls .git`

- `git help`

- `git status`

- modify `hello.txt`

- `git add hello.txt`

- `git diff`

- commit, explain commit, explain that good commit messages are important

- show the hash of the commit

- `git log`

- `git commit -a` commits all changes to all files that are already tracked by git

- but often you don't want to commit everything at once. Reasons for that are

    - Don't want too much changes in one commit

- better: `git log --al --graph --decorate`

BREAK: Show Attractor Project git

For the others to see how this will look for a project

Moving around in the version history

- `git checkout ...` check out previous commit, show that files have changed
- show 'git log –all –graph –decorate" again to see where we are in the graph
- `git checkout master` to go back to the top of the graph
- show what happens if you modify `hello.txt` and then try to `git checkout`
- `git checkout -f`, but BE CAREFUL! Whenever you say anything with `-f` you should really know what you are doing
- `git diff` gives a diff with respect to `HEAD`
- `git diff hash hello.txt` for how it has changed with respect to that hash
- `git diff hash1 hash2 hello.txt` compare two commits
- have `hello.txt` modified –> `git checkout hello.txt` throws away changes in `hello.txt`

Little Python Example: `Animal.py`

To show how one works in a project (commits) and how branching and merging works. I am not sure if this is too much and we should focus on Markus' Mathematica Example

```python
def default():
    print("Hello")

def main():
    default()

if __name__ == '__main__':
    main()
```

- `python animal.py`
- `git status`
- `git add`
- `git commit` write good commit messages!

Git Branches

- `git branch cat` creates branch `cat`
- `git checkout cat` switches to branch `cat`
- could also have done `git checkout -b cat` for both in one command
- add cat function

```python
def cat():
    print('Meow!')
```

- and in main:

```python
def main():
    if sys.argv[1] == 'cat':
        cat()
    else:
        default()
```

- show `git diff`
- run `python animals cat`
- add, commit, show log
- also try `git log --all --graph --decorate --oneline`
- show `git checkout master` (cat stuff gone)
- create dog branch `git checkout -b dog`

```python
def dog():
    print('Woof!')
```

- and in main change it to

```python
def main():
    if sys.argv[1] == 'dog'
        dog()
    else:
        defailt
```

- show git diff
- git add, commit
- `git log --all --graph --decorate --oneline` to see branches

Mergin and Merge Conflicts

- Merge and branch are kind of opposites
- `git checkout master`
- `git merge cat`
- `git log`
- `git merge dog` –> Merge conflict! (in the main function)
- can `git merge --abort`
- `git mergetool` or manually look at the `animal.py`

- if you manually look at it there is going to be `<<<<<<<<<<<<<<`, `=============` and `>>>>>>>>>>>` `dog` in there
  - just fix this stuff

```python
def main():
    if sys.argv [1] == 'cat':
        cat()
    elif sys.argv[1] == 'dog':
        dog()
    else:
        default()
```

- In this example the merge was easy. Usually one should not work on the same part of the code at the same time.
- This is also why modular code is important. If you can just add another feature without having to change anything or very much in the main code than it's easy to collaborate!
- `git merge --continue`
- `git log --all --graph --decorate --oneline`

Other Usefull Commands

- `git add -p animal.py` to stage only some changes in the file

- then `git diff --cached`

- then `git commit`, `git checkout animal.py` to commit and throw away the changes we didn't take

- `git blame` who edited a file

- `git show <hash>` to show the commit diff for this commit (I think)

- `git stash` reverts working directory to last commit

- `git stash pop` will undo the stash

- `git bisect` manually search history. At one point did this test stop working? Complicated stuff.

- `.gitignore` to ignore e.g. `.DS_Store` (Mac) or other stuff (helper stuff in LaTeX)

- Try the Pro Git book

Git Remotes

- Markus