Assignment 8. Change management spelunking and implementation

Useful pointers

- Scott Chacon, Pro Git, 2nd ed. (2014), §§2, 3, 7.1–7.8, 7.11, 10.1–10.5.

Laboratory: Git spelunking

Consider the Git repository and working files in the directory `~eggert/src/gnu/emacs-CS-35L` on the SEASnet GNU/Linux hosts. For this repository, answer the following questions, using Git commands and/or scripts of your own devising. For each question, record which commands you used; if you wrote and used a script, include a copy of the script.

1. How much disk space is used to represent the working files in this directory? How much is used to represent the Git repository? What file or files consume most of the repository space and why?
2. How many branches are local to the repository? How many are remote?
3. How many repositories are remote to this repository, and where are they?
4. What are the ten local branches most recently committed to? List their names in order of commit date, most recent first.
5. How many commits are in the master branch?
6. What percentage of the commits that are in any branch, are also in the master branch?
7. Which ten people authored the most master-branch commits that were committed in the year 2013 or later, and how many commits did each of them author during that period?
8. Use the `gitk` command to visualize the commit graph in this repository. If you are SSHing into SEASnet, you'll need to log in via `ssh -X` or (less securely) `ssh -Y`. Draw a diagram relating the following commits to each other, and explain what likely happened to cause their commit-graph neighborhood. You need not list every single intervening commit individually; you can simply use ellipses.

```
4ea37c2b8b0c5a68fde59770c3536195e0972217
977cd6cb28a37744966ec62f70cf62659f6f302a
625cee531623feddbe3174fad52c7db96ec60bb3
5490ccc5ebf39759dfd084bbd31f464701a3e775
0c06b93c1e467debd401eb0b3be4652fde14fa95
820739bbb572b30b6ce45756c9960e48dca859af
00e4e3e9d273a193620c3a4bb4914e555cb8e343
49cd561dc62ea6b3fbedab7aef0f020733f4cf09
abcb2e62dae6aa26308f7ac9efc89247f89cbe65
98ac36efe4ce4bd3a0bca76fc73ce6c7abaa4371
```

As usual, keep a log in the file `lab8.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened. Your lab note should contain the abovementioned record of the scripts you used.

Homework: Topologically ordered commits

Given a Git repository with multiple branches, the commits can be thought of as having the structure of a rooted tree. In particular, starting from the head of each branch, i.e. the last commit of each branch, one can trace through the commits by following each commit's parent. Once this procedure is done for all the branches, we will get a commit graph, which is a rooted tree, with the root being the oldest commit which we assume all branches share.

Note that if a commit is a merge commit, it will have two parents. But one should always select the parent which gives the longest sequence of commits when tracing up to the root commit. The previous sentence will make sense after you investigate how a merge commit is created.

Follow these five steps to implement `topo_order_commits.py`.

1. Discover the `.git` directory. Inside a directory, when the script `topo_order_commits.py` (which doesn't have to reside in the same directory) is invoked, the script should first determine where the top level Git directory is. The top level Git directory is the one containing the `.git` directory. One can do this by looking for `.git` in the current directory, and if it doesn't exist search the parent directory, etc. Output a diagnostic '`Not inside a Git repository`' to standard error and exit with status 1 if `.git` cannot be found when the search went all the way to the `/` directory.
2. Get the list of local branch names. Figure out what the the different directories inside `.git` do, particularly the refs and objects directories. Read §10.2 Git Internals – Git Objects and §10.3 Git Internals – Git References. One can use the `zlib` library in Python

to decompress Git objects. To simplify this assignment, you can assume that the repository does not use packfiles (see §10.4 Git Internals – Packfiles), i.e., that all its objects are loose.

3. Build the commit tree. Each commit can be represented as an instance of the CommitNode class, which you can define as follows:

```
class CommitNode:
    def __init__(self, commit_hash, associated_branches):
        """
        :type commit_hash: str
        :param associated_branches: the branch names associated with the commit, set of str
        :type associated_branches: set
        """
        self.commit_hash = commit_hash
        self.associated_branches = associated_branches
        self.parents = set()
        self.children = set()

    def __str__(self):
        return f'commit_hash: {self.commit_hash}, associated_branches: {self.associated_branches}'

    def __repr__(self):
        return f'CommitNode<commit_hash: {self.commit_hash}, associated_branches: {self.associated_branches}>'
```

   The commit tree consists of all the commit nodes from all the branches. For a commit object with two parents, figure out which parent to use. Note that all branches should trace to a single root commit. If this assumption is violated, output a diagnostic '`multiple commit roots found`' to standard error, and exit with status 1.

4. Generate a topological ordering of the commits in the tree. A topological ordering for our case would be a total ordering of the commit nodes such that all the ancestral commit nodes come before the descendent commits, where the root commit node is the oldest ancestor. One way to generate a topological ordering is to use a depth-first search; see Topological sorting.

5. Given the ordering in the previous step, print the commit hashes in the reverse order. That is, the the root commit hash should be printed last. When jumping to a sibling branch of the tree, that is, if the next commit to be printed is not the parent of the current commit, insert a "sticky end" followed by an empty line before printing the next commit. The "sticky end" will be the commit hash of the parent of the current commit, with an equal sign "=" appended to it. This is to inform us how the fragments of the tree can be "glued" together to form the tree.

   Furthermore, if a commit corresponds to a branch head or heads, the branch names are listed after the commit in lexicographical order, separated by a white space.

For example, consider the following commits where c3 is a child of c1, and c5 is a child of c4:

```
c0 -> c1 -> c2 (branch-1)
       \
        c3 -> c4 (branch-2, branch-5)
                \
                 c5 (branch-3)
```

A valid topological ordering will be (c0, c1, c2, c3, c4, c5), which should give the following output (assuming the commit hash for cX is hX):

```
h5 branch-3
h4 branch-2 branch-5
h3
h1=

h2 branch-1
h1
h0
```

A equally valid topological ordering will be (c0, c1, c3, c4, c5, c2), which should give the following output:

```
h2 branch-1
h1=

h5 branch-3
h4 branch-2 branch-5
h3
h1
h0
```

As a concrete example, given the repository in the flow-test-dir compressed tarball, the output of `topo_order_commits.py` could be as follows:

```
3c25412d2521b8f77778de839acb0350492c3634 b11 b12
5860da5c2f6181541566190b2f704da20e4753bc
b5d29544d6c69bc654588fbf22071bdaadbc0d23
```

```
777e41d9a9ba12b9a485300e16fa4405a4f633c0=

dd9515b11ebcab015a8a91b494097d06827df5a5 b1
55bb5940a549a475c768b503cb1b76a6f029d444
127169a28b68c1f09a6f28e289b7aafafe3105a3=

0e7b5db7a6ebfe3fa72598d0d7919d64bcfc2ab7 b9
777e41d9a9ba12b9a485300e16fa4405a4f633c0=

59961945fd47abdcf8d526154befb24810fa9a79 HEAD b3 b4 z
5fe5a724533a4970c19fa7bc0986c1f646a07e4c b2
831c585eaada1f7fb3b9f2751b8019dbccc59d94
127169a28b68c1f09a6f28e289b7aafafe3105a3 b8
f72697fe0dbf5455b61ab586346578710fbfb6fd
5e6f9ac703e879de3134990765d2e5dee7ee7ed1
777e41d9a9ba12b9a485300e16fa4405a4f633c0 b10 b13 master
6ad43a64e81fff1fd81de5d2fb6a7c99a71f60a9
a109cb0e9f18fd0f0106216fccb9c9719e5119db
```

Submit

Submit two files:

- The file `lab8.txt` as described in the lab.
- The file `topo_order_commits.py` as described in the homework.

The `.txt` file should be an ASCII text file, with no carriage returns.

$Id: assign8.html,v 1.47 2020/01/06 18:03:00 eggert Exp $