Lesson 1: What version control is, and how `git` compares to other systems.

- Finding difference between two files old.txt and new.txt in Mac and Linux (not sure?): diff -u old.txt new.txt

  What I was using before is opendiff old.txt and new.txt

  This is more involved and outputs a good visual appearance.

- How did viewing a diff between two versions of a file help you see the bug that was introduced? It shows with + and - signs.
- How could having easy access to the entire history of a file make you a more efficient programmer in the long term?
  It certainly helps to debug the codes and reverts some changes which we found later no longer useful.
- How would you prefer to save versions?
  I would prefer it when I would choose to save the version myself.
- How to pronounce Linus Torvalds and Linux?
  Pronounce Leenas Torvalds and linux (not lainux) according to Linux Torvalds
- Some similarities and differences between SVN and git? Manual save - when to save a version at the discretion of the user: both support it. User created checkpoints in git terminology is called **commits**. *Provide a useful comment each time a commit is made.* Use offline - Git is, but SVN isn't.
- What do you think are the pros and cons of manually choosing when to create a commit, like you do in Git, vs having versions automatically saved, like Google Docs does?

  Pros

  ```
  - We can chose when to save, that when a _logical_ change is finished.
  - Code will be compilable for each commits.
  ```

  Cons

  ```
  - One may forget to commit sometimes, and all the version control system fails.
  - Some manual commits may be too big or some may be too small.
  ```

- `git log --stat` to get the list of commits, the changed files, and the number of changes
- `git log -n1` to show only the last commit.
- `git diff <commit no 1> <commit no 2>` to compare to commits.
- Why do you think some version control systems, like Git, allow saving multiple files in one commit, while others, like Google Docs, treat each file separately?

When multiple files are changed, and we committed the *repository* (which is the project or collection of files), git commits all the changed files into the version control system.

- Check git version: `git --version`.
- Using `git log` and `git diff`: to search for a committed message, type `/text` where `text` being the text to be located in committed message. To introduce colored changes in `git diff` type `git config --global color.ui auto`.
- How can you use the commands git log and git diff to view the history of files?
  `git log` will list all the commits with the comments. Search for a commit with a comment, copy its ID, and find the ID of the immediate previous commit. Use `git diff` to find the difference.
- `git clone` will copy all the versions of a repository - a collection of files.

- `git checkout <commit_id>` will revert the present version of the repository to the commit_id version. Now when we issue `git log`, you will see this commit_id in the first line, not the latest commit. To go back to the latest commit, issue `git checkout` again. But always trying remember the master commit_id is hard, may be there's a work around.
- How might using version control make you more confident to make changes that could break something?
  I can revert to any changes later at any point of time.
- Setup the git workspace in terminal.
  - Enable color prompt
  - Enable commit ID number
  - Show an asterisk when editing a checkout
  - Configure an editor with git. I did for
- Now that you have your workspace set up, what do you want to try using Git for?
  On the node age problem.
- `git` commands learned in this Lesson:
  - `git log`
  - `git diff`
  - `git checkout`
  - `git clone`