git basics - Labs

Learning-Catalogue code: 067689

Sébastien SNAIDERO

Software product owner
& senior developer
@ TDS / PDK

life.augmented

Highlights

- Lab 0 - Git & Training Setup
- Lab 1 - Using Git with Local Repo
- Lab 2 - Branching & Merging
- Lab 3 - Solving Merge Conflicts
- Lab 4 - Using Git with Distant Repo
- Lab 5 - Rebasing
- Lab 6 - Push to remote with rebase
- Lab 7 - Rewrite local history (squash)

Lab 0

Git & Training Setup

life.augmented

Lab 0
Git & Training Setup

1. Install Git if necessary
   - Windows http://git-scm.com/download/win
2. Open a terminal with Git loaded
   - Windows Git Bash
   - Linux in a terminal `sw git 2.18.0` (last available version as of writing)
3. Git minimal configuration
   - Set your identity
     - `git config --global user.name "<First> <Last>"`
     - `git config --global user.email <first>.<last>@st.com`
4. Create your ***Training Directory***
     - `mkdir ~/git-training`

Git & Training Setup

4. List your configuration parameters

- `git config --list` ➔ All your settings (last occurence considered)
- `git config --list --show-origin` ➔ All your settings (with origin file)
- `git config [--get] <key>` ➔ Value retained by Git for this <key>

Note : `--show-origin` available only since 2.8.0

5. Some more Git configuration items

- Choose your text editor
  - `git config --get core.editor` ➔ Your current editor (defaults to vi with Unix)
  - `git config --global core.editor '<pathToEditor>'`

```
('"C:\\Program Files (x86)\\Notepad++\\notepad++.exe" -multiInst -notabbar -nosession -noPlugin')
```

- Don't consider trailing spaces (both line & file) as diff conflicts
  - `git config --global core.whitespace -trailing`

Lab 1

Using Git with Local Repo

life.augmented

Local repo - Initialization

1. Go to the ***Training Directory***
2. Create `lab1` directory and `cd` into it
3. Use git command to initialize a git repository in that directory :

   ```
   git init
   ```

   - Observe that a .git directory is created
4. Git configuration
   - List all configuration parameters
     - `git config --list --local` ➔ Should return something

Local repo - First commit

1. Create `file1.txt`
   - Observe the output of `git status`
     - file1.txt must be in the ***untracked*** area
   - Observe also the help proposed by `git status`
2. Add the file to the staged area
   - Use `git add` to add the file
   - Use `git status` command to confirm the staging success
3. Commit the content of the staged area
   - Use the `git commit` command
   - Observe the commit creation message & all the information it provides to you
     - SHA-1 checksum of the new commit
     - How many files were changed
     - Statistics about lines added and remove
     - ...
   - Check `git status` to confirm no more local changes exists

Local repo - Commit updates

1. Update `file1.txt`
   - Observe the changes with `git diff`
   - Check working repository situation with `git status`
     - file1.txt is modified and not staged
2. Add the file to the staged area
   - confirm using the `git status` command
3. Commit the modifications
   - Check the history of your commits with `git log`
4. Modify again `file1.txt` and add it to the staged area
   - Use `git commit --amend` to append the modification to the last commit
   - Check the evolution of the history of your commits with `git log`
5. Use `git show` to see the details of a commit

Local repo - Undo & Discard

1. Create updates
   - Modify file1.txt then `git add`, then modify file1.txt again
   - Observe the result of `git status`
2. Discard changes in working directory
   - Undo with `git checkout -- file1.txt` & check status and file content
3. Unstage changes
   - Use the command `git reset` to unstage the file & check status and file content
3. Undo last commit, keep changes in working dir
   - Commit current changes in `file1.txt` & check the history
   - Use `git reset HEAD~1`, check status & check the history
4. Discard last commit
   - Commit changes from previous step & check the history
     - same commit content, same author, different SHA1
   - Use `git reset --hard HEAD~`, check status & check the history

Lab 2

Branching & Merging

life.augmented

Branching - Create branch

1. From ***Training directory***, create a new local repo named `lab2`
   - Create and commit two files (file_1.txt & file_2.txt) in branch ***master***
2. Create a new branch named ***lab-merge***
   - Use `git branch lab-merge` to create the branch
   - Use `git branch` to list all your available branches
3. Switch to the new branch
   - Use `git checkout` command
   - Use `git status` command to confirm the switch
4. Create two new commits including the following activity
   - Modify file_2.txt, then commit
   - Create file_3.txt, then commit

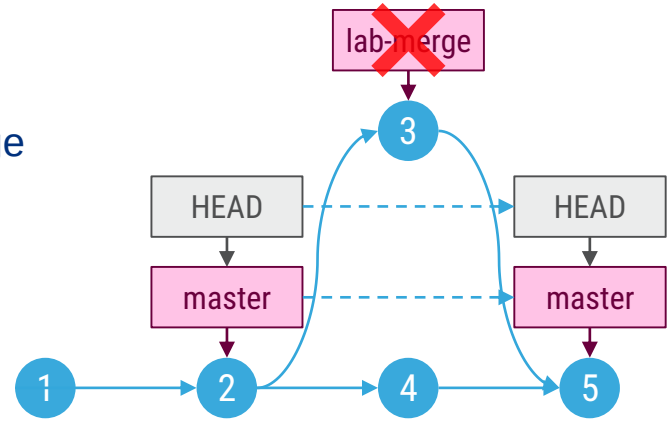
Merging - Fast Forward

1. Switch back to ***master*** branch
2. Merge the ***lab-merge*** branch into ***master***
   - Use `git merge [--ff] lab-merge` to merge the branch into master
   - Observe the **Fast-forward** merge message
   - Confirm the merge success using the `git log` command
   - You can also observe the merge success using `gitk` command
3. Check that working directory is clean
   - Use `git status` command to ensure

Lab 2

Merging - With Commit

1. Switch to ***lab-merge*** branch
   - Use `git status` command to confirm the switch
2. Create a new commit including the following activity
   - Modify file_2.txt, then commit
3. Switch back to ***master*** branch
4. Create a new commit including the following activity
   - Modify file_1.txt, then commit
5. Merge ***lab-merge*** branch into ***master***
   - The editor is opened: enter merge commit message
   - Observe the merge commit creation
6. Delete the ***lab-merge*** branch
   - Use `git branch -d lab-merge` command
   - Confirm with `git branch`



Lab 3

Solving Merge Conflicts

Merging - Create Conflict

1. From ***Training directory***, copy `lab2` to `lab3`
2. Create a new branch named ***lab-conflicts*** in your ***lab3*** local repo
   - Use `git checkout -b lab-conflicts` command
   - Use `git status` command to confirm branch creation and switch
3. Create two new commit including the following activity
   - Modify file_1.txt, then commit
   - Modify file_1.txt again, then commit
4. Switch back to ***master*** branch
5. Create one new commit including the following activity
   - Modify file_1.txt, then commit
5. Merge the ***lab-conflicts*** into ***master***
   - Notice the merge conflict message
   - Use `git status` command to see the files concerned by the merge conflict

Merging - Solve Conflict

6. Resolve the conflict
   - Edit file_1.txt & find the conflict

```
<<<<<<< HEAD
... HEAD branch code ...
=======
... Merged branch code ...
>>>>>>> merged-branch
```

   - Fix the conflict (i.e. select what to keep) then save
7. Confirm the merge conflict resolution
8. Check the merge status
   - Notice the message: all conflict fixed but you are still merging
9. Finalize the merge
   - Notice that conflict resolution is done through a new commit : the merge commit
10. Confirm the merge & delete ***lab-conflict*** branch

Lab 4

Using Git with Distant Repo

life.augmented

Distant repo - Initialization

1. Be sure to have set your SSH key as explained in the FAQ
2. Go to the ***Training Directory***
3. Use `git clone` command to clone the remote repository provided by the trainer as `lab4`
4. Move into the new cloned project
5. Check the distant repo associated using `git remote -v`
6. Explore the history with `git log`

Distant repo - Commit updates

1. Create a new commit including following activity
   - Into `users/` folder, create a folder with your name
     - users/<yourName>
   - Create two files (file_1.txt & file_2.txt) into that folder
     - Use `git status` command to check working directory information
   - Use `git add` command to add your folder (use the folder name)
     - Use `git status` command to notice that all the folder content is staged
   - Commit with following message
     - “<yourName>: add files 1 & 2”
     - Use `git log` to check commit is done

2. Synchronize with remote to update your local repo
   - `git pull`
   - Use `git log` command to observe the local repo updates ➔ very verbose
   - `git log --graph --oneline --all` ➔ better ?
   - Create an alias for this command & test it
     - `git config --global alias.lg "log --graph --oneline --all"`
     - `git lg` ➔ same result ?

3. Push your commit to the remote
   - `git push`
   - Confirm using `git lg` or `gitk --all` to test the built-in GUI

Lab 5

Rebasing

life.augmented

Rebase & Solve conflict

1. From ***Training directory***, git clone `lab3` as `lab5`
   - Check that remote `origin` is associated with lab3 using `git remote -v`
2. Create & switch to a new branch named ***lab-rebase*** from ***master*** in your ***lab5*** repo
3. Create two new commits including the following activity
   - Modify file_1.txt, then commit
   - Modify file_1.txt again, then commit
4. Switch back to ***master*** branch and create one commit:
   - Update file_1.txt to create a conflict and commit
5. Rebase ***lab-rebase*** against ***master***
   - Checkout ***lab-rebase*** branch
   - Rebase against ***master*** using `git rebase master`
   - View rebase operation effect in history

Lab 6

Push to remote with rebase

life.augmented

Distant repo - Add remote & rebase

1. Create a new local git repo from ***Training directory***, named `lab6`
2. Add new remote named ***codex*** from remote provided by trainer
   - Use `git remote add codex <remote>` command
   - Use `git remote -v` to confirm that the remote is added
   - Use `git fetch codex` to synchronize remote repository content
   - Explore `git pull` command to checkout the `master` branch
3. Do the same job that in the step 'Create a new commit including following activity' of ***Lab 4***
4. Try to push to the remote
   - try to push your changes using `git push`
   - on error, use `git pull --rebase`
   - push your update using `git push`
5. Compare the final structure of the history with Lab4

Lab 7

Rewrite local history

Lab 7
Squash

1. Clone the remote repository provided by the trainer as `lab7` & move into
   - You should get the following history

```
* 57ae65a (HEAD -> master, origin/master) add perimeter computation
* dd06ab2 style: fix indentation
* b826ebc add area computation + fix missing semicolons
* 2aaa3fe fix constructor method name
* e9b1213 add new class Rectangle
* 128cd9e add file class.js
```

2. Rebase using `git rebase -i 128cd9e`, so that
   - `e9b1213` is kept
   - `2aaa3fe` is squashed into `e9b1213`
   - `dd06ab2` is squashed into `b826ebc`
   - `57ae65a` is kept

3. Use diff to compare `57ae65a` with your last squashed commit

4. Expert mode, redo the exercise with these 2 steps instead
   - edit `b826ebc` to split functional part (1st commit) from style part (2nd commit)
     - Use `git reset`, then `git add -p`
   - `dd06ab2` is squashed in previously created style commit