

LAB #3 - GIT (Advanced)

- ❗ This task needs to be completed using command line git on `enos.itcollege.ee` server (except where indicated). Before you start your lab task, please **fully read** this specification! Because the order of commands is important, each command line command is numbered.
- ❗ Your task consists of 17 major steps. Follow these in the given order!
- ❗ Do not copy-paste from the PDF file. Some characters may convert incorrectly!

Overview

- Step 1 : generate an `ssh` key on `enos.itcollege.ee`.
- Step 2 : add the `ssh` key to gitlab to speed up git use by enabling `ssh` connections.
- Step 3 : make a new empty repository.
- Step 4 : clone the repository with `ssh` link.
- Step 5 : provide user information that can help to identify you and to contact you.
- Step 6 : create an empty file on command line and add it to your remote repository.
- Step 7 : add tags for versioning and rollback backups.
- Step 8 : add branches for feature testing and development before merging with **master** branch and creating a tag for version/milestone.
- Step 9 : switch between branches.
- Step 10 : edit/modify files from command line.
- Step 11 : restore a branch that was mistakenly/accidentally deleted from both local and remote repository.
- Step 12 : revert an erroneous commit, i.e., undo changes that were pushed to remote repository.
- Step 13 : merge two branches that do not have modifications/edits in the same files, i.e., merge without a conflict; also save it as a milestone/version/tag.
- Step 14 : work on a new feature that involves editing files that are edited also in other branches.
- Step 15 : resolve merge conflicts.
- Step 16 : save the commands used for the lab for backup.
- Step 17 : (optional) work on a project with other team members; resolve merge conflicts; access branches made by other team members.

Margin notes contain ❗ and ✅ symbols. ❗ indicates that the line/paragraph is of uttermost importance and ✅ indicates that the line/paragraph contains something interesting or useful but not strictly mandatory.

Step 1 – Generate an ‘ssh key’

Log in to `enos.itcollege.ee` to generate an ‘ssh key’:

```
100 ssh-keygen -t ed25519 -C "enos.itcollege.ee" # generate an ssh key
```

Use default values, i.e. press ENTER-key when asked for file name and passphrase:

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/rene.pihlak/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rene.pihlak/.ssh/id_ed25519
Your public key has been saved in /home/rene.pihlak/.ssh/id_ed25519.pub
...
```

You can view the **public** `ssh` key with the following command:

```
101 cat ~/.ssh/id_ed25519.pub # display contents of the public key file
```

The result should look something similar to:

```
ssh-ed25519 RAnD0MlEtTErAnDnuMBerS enos.itcollege.ee
```

- ❗ For security reasons keep this key secret, do not share it with others!
- ✅ Read the guide [How to generate SSH Keys](#) to set up your home computer too.

Step 2 explains how to enable `ssh` connections to gitlab.

Step 2 – Add 'ssh key' to gitlab profile

You can add multiple ssh keys to your gitlab profile. In this example, you are required to add the ssh key that you generated on `enos.itcollege.ee`.

Log in to gitlab.cs.ttu.ee. Next click on your profile picture and then Settings > SSH Keys. Paste the ssh key in the text box. The Title is generated automatically. **Do not set 'Expires at'!**

To finish click on **Add key** button (see Figure 1).

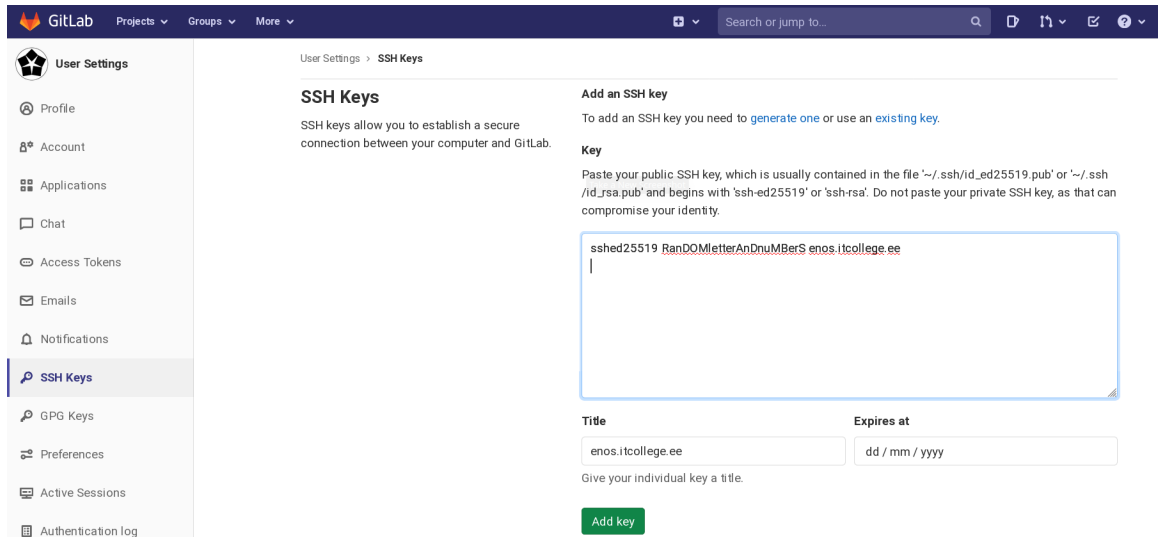


Figure 1: Add SSH key

Now you should be able to clone your gitlab repositories with ssh instead of https. This will allow you to do `git push` and `git pull` without having to type your username and password every time.

Step 3 – Create a new empty repository

In gitlab create a new repository **without README** (see Figure 2). This is a new repository, not a folder inside the repository for other labs. Name the repository `icd0007_lab3_yourname` where `yourname` is your Uni-ID username. Make sure you add me as a member to your repository.

We will create all files and folders from command line in `enos.itcollege.ee` server!

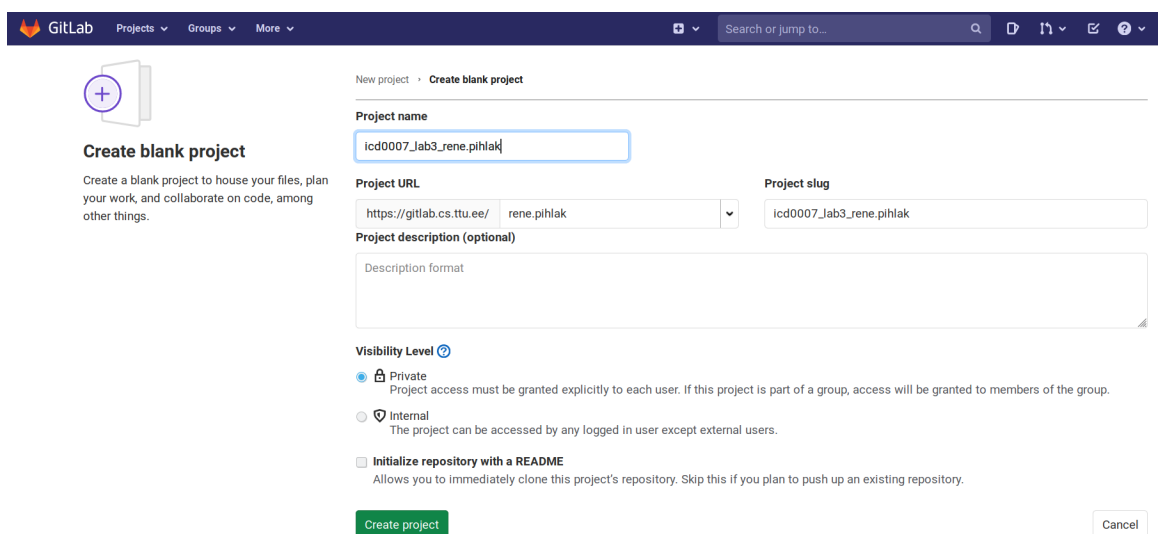


Figure 2: Create new repository for Lab#3

Step 4 – Clone the empty repository using ssh

Copy the address to clipboard. Choose the address under '**Clone with SSH**' (see Figure 3).

- ❗ **From now on, for this lab, you are not permitted to use the gitlab web interface for any git commands! Only use gitlab to verify that your commits, etc appear in the repository.**

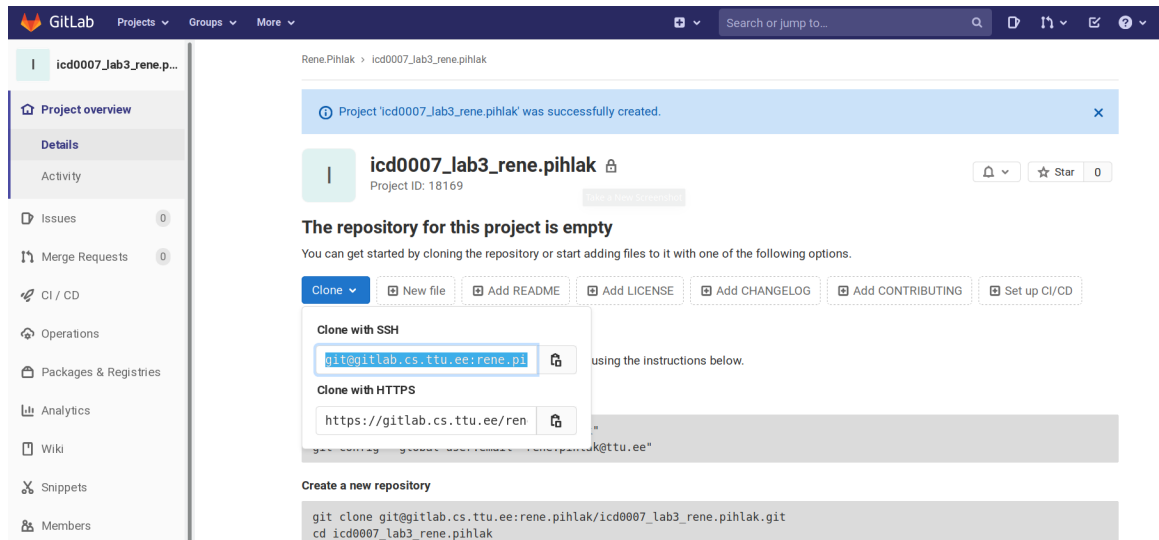


Figure 3: Get the ssh address for cloning

Go to `enos.itcollege.ee` and clone the repository you just made. When asked if you want to continue connecting, answer **yes**.

- ❗

```
102 echo "BEGINNING" # set a starting point for history: IMPORTANT!! See Step 16
103 git clone git@gitlab.cs.ttu.ee:rene.pihlak/icd0007_lab3_rene.pihlak.git # clone repo
```

You are informed that you are cloning an empty repository. Keep calm and continue. Change your working directory to the folder that was created by cloning:

```
104 cd icd0007_lab3_rene.pihlak/ # change directory
```

Step 5 – Configure git user

If you have not configured your global git user, then configure it with your data now before continuing with the rest of the task(s).

```
105 git config --global user.name "René" # use your name
106 git config --global user.email "rene.pihlak@taltech.ee" # use your e-mail
```

In case you prefer to configure each git folder separately, use the above commands while omitting `--global`.

Step 6 – Add an empty file to your repository

Next add an empty `README.md` file and push it to the gitlab repository.

```
107 touch README.md # create an empty README.md file
108 git add -A # add all changes
109 git commit -m "init README.md" # create commit message
110 git push # push changes to current branch (master)
```

If you refresh your browser window in gitlab, you should see that a new file was added.

Step 7 – Add a 'tag' to your repository

Let's add a tag to mark that this is our clean base:

```
111 git tag -a v0.0 -m "base version" # make a local tag "v0.0"
112 git push origin v0.0 # add this tag to remote repository in gitlab
```

Now if you click on the **master** pull down button you should see that you have a branch called **master** and a tag called **v0.0** (see Figure 4).

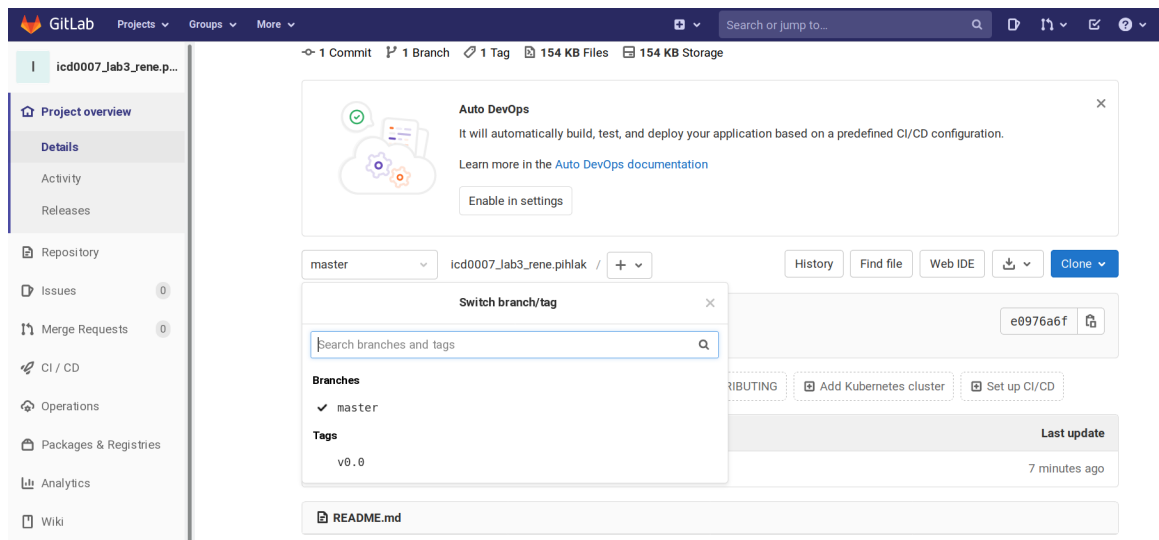


Figure 4: First tag

- ✓ Usually tags are used for versioning (therefor, adding v0.0 to an almost empty repository has little practical value). We will add more tags for different milestones later on.

Step 8 – Add a ‘branch’ to your repository

Create a new branch for testing. Create an empty `index.html` file and commit it to your repository on gitlab.

```
113 git checkout -b b_test # this makes a copy of current branch (master)
114 git push -u origin b_test # create a remote branch on gitlab
115 touch index.html # make empty index.html file
116 git add -A # add all changes
117 git commit -m "init index.html" # write your commit message
118 git push # push changes into the new remote branch
119 ls -l # check what files are in the folder
```

In the command line terminal you should see that you have the following files in your folder:

```
README.md
index.HTML
```

Step 9 – Move between branches in your local repository

You can move between branches. To illustrate this, move to **master** branch:

```
120 git checkout master # move to master branch
```

Verify that the **master** branch does not have the `index.html`:

```
121 ls -l # check what files are in the folder
```

You should have only the following file in your folder:

```
README.md
```

Step 10 – Modify files in a new branch

Create a new branch for all the `html` files. Create an empty `index.html` file and commit it to your repository on gitlab.

```
122 git checkout -b b_html # this makes a copy of current branch (master)
123 git push -u origin b_html # create a remote branch on gitlab
124 touch index.html # make empty index.html file
125 git add -A # add all changes
126 git commit -m "init index.html" # write your commit message
127 git push # push changes into the new remote branch
128 ls -l # check what files are in the folder
```

Move back to **master** branch and edit `README.md` with `vim`.

```
129 git checkout master # move to master branch
130 vim README.md
```

In vim press i-key to enter '**INSERT**' mode. Write the name of the lab and your own name as below:

```
# LAB 3
## Author: René Pihlak
```

Press ESC-key to exit **INSERT** mode. To save the changes type the following two letters ':w' and press ENTER-key. To exit vim type the following two letters ':q' and press ENTER-key.

Again, let's commit and push these changes to the remote repository.

```
131 git add -A # add all changes
132 git commit -m "title, author" # write your commit message
133 git push # push changes into the remote master branch
```

The same way you could edit the README.md in other branches. In Step we learn how to add this modification 'automatically' to another branch.

Step 11 – Restore a branch that was accidentally deleted from the repository

Let's 'accidentally' delete the branch b_html from both local and remote repository:

```
134 git checkout master # make sure we are in master branch
135 git branch -d b_html # DELETE local branch
136 git push -u origin :b_html # DELETE a remote branch on gitlab
137 git branch -vv # list branches
```

You should see something like this:

```
b_test cb17e5d [origin/b_test] init index.html
* master 3117e7b [origin/master] title, author
```

❗ The hash codes may differ in your case. This means that the local branch is deleted. You will see on gitlab that the remote branch b_html is deleted too! But not all is lost. Let's revert our mistake.

First, let's find the last valid commit for branch b_html.

```
138 git reflog # find the last valid commit to b_html
```

This should give something like this (again hash codes may differ):

```
3117e7b (HEAD -> master, origin/master) HEAD@{0}: checkout: moving from master to master
3117e7b (HEAD -> master, origin/master) HEAD@{1}: commit: title, author
b70dc58 (tag: v0.0) HEAD@{2}: checkout: moving from b_html to master
7c949a1 HEAD@{3}: commit: init index.html
b70dc58 (tag: v0.0) HEAD@{4}: checkout: moving from master to b_html
b70dc58 (tag: v0.0) HEAD@{5}: checkout: moving from b_test to master
cb17e5d (origin/b_test, b_test) HEAD@{6}: commit: init index.html
b70dc58 (tag: v0.0) HEAD@{7}: checkout: moving from master to b_test
b70dc58 (tag: v0.0) HEAD@{8}: commit (initial): init README.md
```

You may notice that one of the lines does not have '(...)' before 'HEAD@'. This is because that branch was deleted. If this is not clear (because there are other branches that we have correctly deleted), then you should do your best to recall the last commit's message. In our case, the last commit to branch **b_html** was made with commit message 'init index.html':

```
7c949a1 HEAD@{3}: commit: init index.html
```

❗ **This is why it is important to have informative commit messages!**

Let's re-create local b_html branch from that commit using the hash code of the commit:

```
139 git checkout -b b_html 7c949a1 # create local branch b_html from old commit
140 ls -l
```

You should see that the index.html file is restored:

```
README.md
index.html
```

Push the local branch back to remote repository:

```
141 git push -u origin b_html # create a remote branch on gitlab
```

Step 12 – Revert accidental commits to the repository

Now that we know how to restore branches, let's try to revert a commit that was performed in mistake. Let's first make the commit and then correct it.

```
142 git checkout b_html # make sure we are in b_html branch
143 touch style.css # create an empty file
144 git add -A # add the new file
145 git commit -m "style.css in wrong branch"
146 git push
```

Because we realized our mistake right away, it can be fixed quickly:

```
147 git revert --no-edit HEAD # undo the last commit
148 git commit -m "Revert: remove style.css"
149 git push
```

Step 13 – Merge two 'simple' branches

Let's merge two 'simple' branches: **master** and **b_html**. The goal is to make the two branches have identical files/folders and to insert the missing commits to branches so that the history of commits is preserved even if a branch is later deleted:

```
150 git checkout b_html # make sure we are in b_html branch
151 git rebase master # notice that we are in b_html branch
152 git pull
```

The last command will most likely open some text editor (nano, vim). If it is vim then type `:wq` and press ENTER-key to save and quit. In case it is opened in nano or pico, press Ctrl-X. Follow instructions if it does not close the editor after that.

Let's visualize the repository and its commits as a graph diagram (in ASCII form):

```
153 git log --graph --oneline # show ASCII graph
```

The ASCII version of the graph is rather primitive:

```
* 07f183a (HEAD -> b_html) Merge branch 'b_html' of gitlab.cs.ttu.ee:rene.pihlak/icd0007_lab3_rene.pihlak into b_html
|\
| * bd13b89 (origin/b_html) Revert "style.css in wrong branch"
| * e91b273 style.css in wrong branch
| * 7c949a1 init index.html
* | 564970b Revert "style.css in wrong branch"
* | fea7568 style.css in wrong branch
* | 4103d14 init index.html
* | 3117e7b (origin/master, master) title, author
|/
* b70dc58 (tag: v0.0) init README.md
```

✔ For a more elegant diagram, take a look in Repository > Graph on gitlab.

You notice that HEAD -> b_html is ahead of origin/b_html and origin/master. Let's synchronize the two branches:

```
154 git push # push changes to remote branch b_html
155 git checkout master # move to master
156 git merge b_html # merge master and b_html
157 git push # push to remote master
158 git log --graph --oneline # show ASCII graph

* 07f183a (HEAD -> master, origin/master, origin/b_html, b_html) Merge branch 'b_html' of gitlab.cs.ttu.ee:rene.pihlak/icd0007_lab3_rene.pihlak into b_html
|\
| * bd13b89 Revert "style.css in wrong branch"
| * e91b273 style.css in wrong branch
| * 7c949a1 init index.html
* | 564970b Revert "style.css in wrong branch"
* | fea7568 style.css in wrong branch
* | 4103d14 init index.html
* | 3117e7b title, author
|/
* b70dc58 (tag: v0.0) init README.md
```

We have now reached our next milestone: project with only `html` file(s), so let's save this stage as a tag:

```
159 git tag -a v0.1 -m "HTML only"
160 git push origin v0.1
```

Step 14 – Work on a new feature in a new branch

Let's make a branch for CSS, a sub-folder and a new file:

```
161 git checkout master
162 git checkout -b b_css # this makes a copy of current branch (master)
163 git push -u origin b_css # create a remote branch on gitlab
164 mkdir styles
165 touch styles/main.css # make empty main.css file
166 git add -A # add all changes
167 git commit -m "init main.css" # write your commit message
168 git push # push changes into the new remote branch
```

Edit `README.md`.

```
169 vim README.md
```

In `vim` press `i`-key to enter '**INSERT**' mode. Add text to the file. Press `ESC`-key to exit **INSERT** mode. To save the changes type the following two letters `':w'` and press `ENTER`-key. To exit `vim` type the following two letters `':q'` and press `ENTER`-key. (Saving and quitting can be combined as `':wq'` and press `ENTER`-key.)

The edited file should look like:

```
# LAB 3
## Author: René Pihlak

Project with
- CSS
```

Push changes to remote branch `b_css`:

```
170 git add -A # add all changes
171 git commit -m "README: with CSS" # commit with message
172 git push
```

Then move to `master` branch and make similar edits to `README.md`:

```
173 git checkout master # move to master branch
174 vim README.md # edit the file
```

In `vim` press `i`-key to enter '**INSERT**' mode. Add text to the file. Press `ESC`-key to exit **INSERT** mode. To save the changes type the following two letters `':w'` and press `ENTER`-key. To exit `vim` type the following two letters `':q'` and press `ENTER`-key. (Saving and quitting can be combined as `':wq'` and press `ENTER`-key.)

The edited file should look like:

```
# LAB 3
## Author: René Pihlak

Project with
- HTML
```

Push changes to remote master branch:

```
175 git add -A # add all changes
176 git commit -m "README: with HTML"
177 git push
```

These two branches have now conflicting edits. Merging these branches is shown in Step 15.

Step 15 – Merging two branches and resolving merge conflict

- ❗ Because `README.md` was edited in two branches, merging these branches will result in merge conflict!

```
178 git checkout b_css
179 git rebase master
180 git diff
```

We can see that there is one line that is different in two branches that we are merging.

```
diff --cc README.md
index 97aa1f7,af69ac2..0000000
--- a/README.md
+++ b/README.md
@@@ -2,4 -2,4 +2,8 @@@
  ## Author: René Pihlak

  Project with
+<<<<<<< HEAD
+- HTML
+=====
+ - CSS
+>>>>>>> README: with CSS
```

Resolve the conflict manually with vimdiff (see Figure 5).

```
181 git mergetool --tool=vimdiff
```

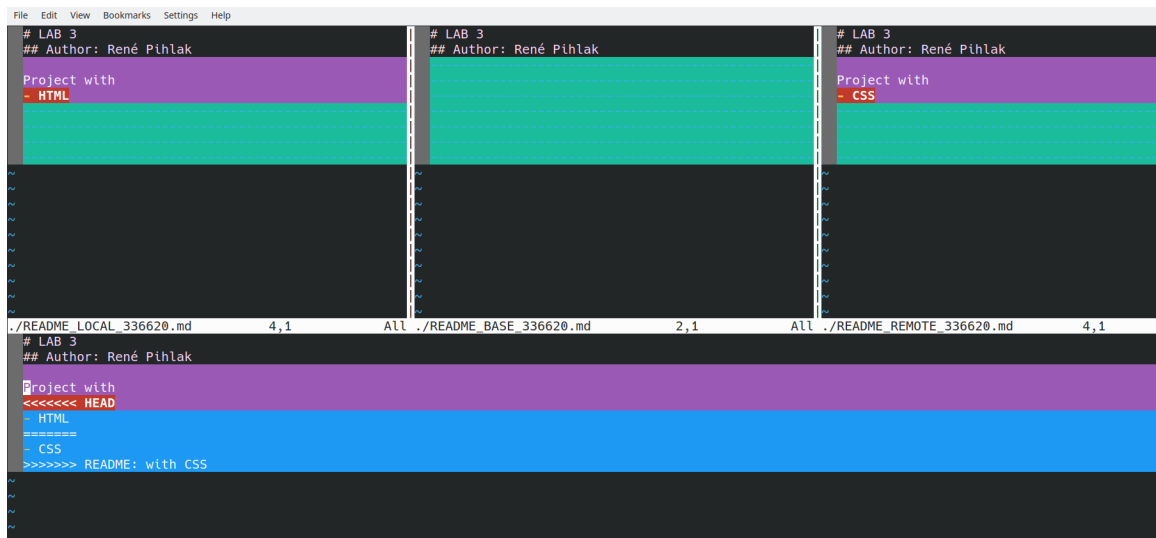


Figure 5: Starting to solve the merge conflict

Navigate with arrow keys. To delete whole lines, we do not need to enter **INSERT** mode. Type two letters 'dd' to delete a whole line under the cursor. The final version should look like in Figure 6.

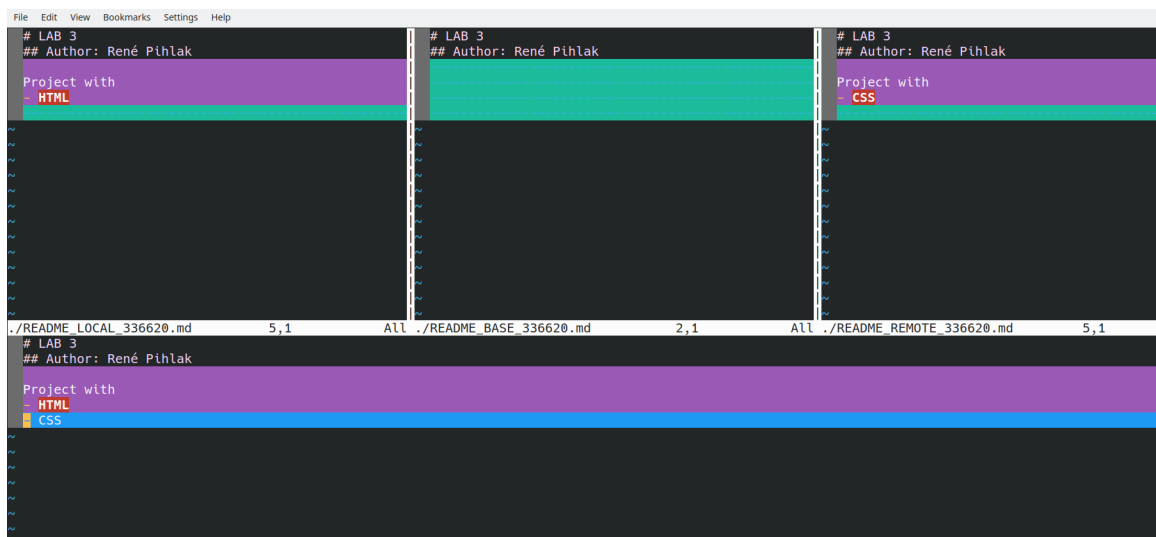


Figure 6: Solved merge conflict

Type two letters ':w' to save the file. Then type three letters ':qa' to close all files.

Let's try to confirm that you have resolved the conflict (lines 182–183). But the remote branch gives a new conflict message (line 184). Edit the README.md again, and push the changes to remote branch.


```

182 git add README.md
183 git rebase --continue
184 git pull
185 vim README.md
186 git add README.md
187 git commit -m "README: resolved"
188 git push

```

- ✔ (There are vimdiff commands what can speed up the selection of correct edits, but this is left for the students to discover.)

Fast forward the master branch; update the remote branch and add a milestone tag:

```

189 git checkout master
190 git merge origin/b_css
191 git push
192 git tag -a v1.0 -m "HTML+CSS"
193 git push origin v1.0
194 git log --graph --oneline

```

```

* 85bb6c1 (HEAD -> master, tag: v1.0, origin/master, origin/b_css, origin/HEAD, b_css) README:
  resolved
|\
| * 589166b README: with CSS
| * 0688d53 init main.css
* | 798a3b1 README: with CSS
* | 1e6de46 init main.css
* | 1a90579 README: with HTML
|/
* 07f183a (tag: v0.1, origin/b_html) Merge branch 'b_html' of gitlab.cs.ttu.ee:rene.pihlak/
  icd0007_lab3_rene.pihlak into b_html
|\
| * bd13b89 Revert "style.css in wrong branch"
| * e91b273 style.css in wrong branch
| * 7c949a1 init index.html
* | 564970b Revert "style.css in wrong branch"
* | fea7568 style.css in wrong branch
* | 4103d14 init index.html
* | 3117e7b title, author
|/
* b70dc58 (tag: v0.0) init README.md

```

Step 16 – Add command history

- ❗ Make sure that you recorded the starting point of git commands in Task#2:

```

102 echo "BEGINNING" # set a starting point for history

```

Save the commands you used to solve the lab into a file history.txt:

```

195 history -w # write used commands to ~/.bash_history
196 awk '/BEGINNING/{y=1; next}y' ~/.bash_history > history.txt # from BEGINNING
197 git add -A # add new file
198 git commit -m "submitting history.txt" # add message
199 git push # update remote repository

```

Step 17 – Work in teams (optional)

Add another team member to your project in git lab. You can add more than one. Using the examples above, make commits to each other's repositories. Create new branches for new features. Work on the same branches and files. Solve merge conflicts.

- ❗ Highly recommended!
- ✔ This will help you learn how to manage real team work. It avoid high blood pressure when the project's deadline is approaching as you would be more confident how to resolve 'git' related issues.

For example, each member should clone and checkout each other's projects at version v1.0. Add your name and only your own name to the README.md and push it to the **master** branch after others have made their commits to **master** branch. Make sure you are the last person to push your commits. Resolve conflicts.