Exercise 1 - prepare basic git environment

git clone https://github.com/lsnp/slideshow\_.git

1. Prepare 'slideshow' project:

a) clone it from the link provided by the trainer

(git clone)

- remove '.git' folder from 'slideshow\_'

(cd slideshow\_; rm -rf .git)

b) initialize project in 'slideshow\_'

(git init, ls -la, git add, git commit)

c) create folders for all of your git projects

and make bare project called 'slideshow.git'

(~/repos, cd ~/repos, git clone --bare)

d) create first working copy of 'slideshow.git', name it 'wc1'

(git clone)

e) check status in both repos, 'wc1' and 'slideshow.git',

and analyze messages - what can you tell?

(git status, ls)

Exercise 2 - more working copies

2. Create another working copy 'wc2' from bare repo

you did in the first exercise

(git clone)

Exercise 3 - daily routine

3. In 'wc1':

a) make some order with folders

- move all pictures into new folder (pic)

(use 'mv' and check the repo with 'status' and 'diff')

- move all javascript files into new folder (js)

(use 'git mv' and check the repo)

b) analyze situation - what can you observe? (-:

c) add to cache changes from a)

(git add)

d) check the repo

e) create revision

(git commit)

f) check the repo

g) send changes to remote repo

(git push)

h) check the repo

i) add new file 'logo5.html', copy file 'logo.html' into it

and change type of picture rotation

(in the first row all should have 'left')

j) check the repo (wc1), status, diff

k) send changes to remote repo

l) check the repo again

Exercise 4 - synchronize with others

4. Actualize 'wc2' and analyze messages

(git pull, git status)

Exercise 5 - empty folder, daily routine con't

5. In 'wc2':

a) create new empty catalog 'html'

- check status - what happened?

- send changes to remote repository (how?)

- check status

b) in 'html' folder place all files with extension '.html'

- check status, but don't commit yet

c) create new file "logo\_jq2.js", copy "logo\_jq1.js"

and change parameters (delay, speed, timeout) for rotation

of type 'slideY'

- check the repo and analyze messages

- send changes to remote repository

- check the repo again

d) test your project - what are the conclusions?

Exercise 6 - even more daily routine

6. In 'wc1'

a) cut 'css' code from 'logo.html', place it in new file

"logo.css" and link it:

- check the repo and analyze messages

- send changes to remote repo

b) actualize 'wc1' and analyze messages

c) send changes to remote repo

Exercise 7 - time machine, stage

7. In 'wc2':

a) remove "logo5.html", use just 'rm'

- check the repo and analyze messages

- go back (git checkout)

b) remove "logo5.html" again, but this time use

'git rm'

- check the repo and analyze messages

- go back (git reset, git checkout)

c) finally remove "logo5.html" and send changes to remote repo

d) actualize wc and analyze messages

e) change the table frame, in file "logo.css" add solid rectangle

- check the repo

- send changes to remote repo

- check the repo again

Exercise 8 - am I assertive?

(make sure that all wc's are clean)

8. Provide those changes:

a) in 'wc1':

- actualize repo

- change table frame to dotted in 'logo.css'

- check status

- send changes to remote repo

- check status

b) in 'wc2':

- DO NOT actualize repo (-:

- check status

- change table frame to dashed in 'logo.css'

- send changes to remote repo

- check status

- what can we do now? (--:

Exercise 9 - judge Dredd..

9. Find out who and when created 'logo.css'?

(git annotate, git blame, git log)

Exercise 10 - mirror, mirror who's the prettiest?

10. Check the differences between versions of 'logo.css' (git diff, git show)

a) for one revision

b) between 2 revisions from before this file was created

c) between 2 revisions after it appeared

d) change the order of rev names in c) and do diff again

- do you have any conclusions?

Exercise 11 - going back, stashing

11. Safe way to keep your temporary changes 'out of the view':

a) in 'wc1':

- show pictures in their default size, comment css rule in 'logo.css' ( /\* ... \*/ );

- move changes to remote repo

b) in 'wc2':

- DO NOT actualize it!

- change the width of pictures to '190px'

- do not stage changes yet

- now actualize it

- hide your changes temporarily

(git stash)

- check status (what does git say to you?)

- actualize it again

- reapply temporary changes

(git stash)

- what can/should you do now? (discussion)

- test these ways of 'going back':

-- git checkout <rev>, git checkout master

-- git checkout <rev> <file name>, cat <file name>, get temporary changes from stash

-- git reset HEAD <file name>, git checkout -- <file name>, get last change from stash

-- git revert HEAD

-- git revert HEAD <file name>

- send changes to remote repo (solve the conflict)

Exercise 12 - boring history..

12. Find out what has changed in file "logo\_jq1.js" (git log)

a) in general

b) between 2 revisions (for example 4 and 7)

c) between yesterday and today

d) in short version with:

- 7 first characters of SHA1

- one line

- short comment of revision

12.1. Check in a quick way how many of commits did your user in 'wc1'

(shortlog)

Exercise 13 - Daisy's diary

13. Look at the changes for "logo.css" from reflog (git reflog)

a) in general

b) for concrete branch

c) clean all reflog

Exercise 14 - fb ugliest girls

(Clean 'wc1' and 'wc2')

14. Binary conflict:

a) in 'wc2':

- open gitk (gitk --all &)

- change .png file and stage it

- check the difference between stage and HEAD

- send changes to remote repo

b) in 'wc1' (go there in another one terminal):

- DO NOT actualize it!

- open another gitk (gitk --all &)

- change .png file

- send changes to remote repo

(git checkout --theirs <file>, git add --all, git commit)

Exercise 15 - lazy deploy

15. Build new version of the whole project (git archive)

Exercise 16 - headless chick

16. Use time machine and go back to the 'logo.html' in revision 4.

Check status and go back to the future (-:

Exercise 17 - tree monkeys

(create 'wc3')

17. In 'wc3' create branch 'wc3\_branch1' (git branch, git checkout)

a) in 'wc3\_branch1':

- change 'logo1.html', go back to 'master' and check status, switch back to 'wc3\_branch1'

- stage your changes, go back to 'master' and check status, switch back to 'wc3\_branch1'

- commit changes, go back to 'master' and check status, switch back to 'wc3\_branch1'

- watch changes in gitk (gitk --all &), push changes(?), pull changes(?)

- change height of picture in 'logo.css' to '150px'

- commit changes

- check the history, go back to master and check history again, any conclusions?

b) in 'master' apply changes:

- in 'logo.css' change height to '200px'

- commit changes

- add new row in table in 'logo1.html' and commit this change, watch it in gitk (refresh with F5)

- check the history, send changes to remote repo

- watch it again with gitk (F5)

Exercise 18 - tree monkey con't

18. Combine changes from main branch with 'wc3\_branch1' (pwd, git merge)

18.1. Create 3 new branches

- start them in 3 different revs in master

- at least 2 revs in each,

- and merge them into master in one step

(git merge)

18.2. Create 2 new branches

- first one starts in master

- second one starts in the first one

- and merge them into each other in the reverse order (two steps)

18.3. Pushing branches

http://training-course-material.com/training/Git\_-\_Branching\_and\_Merging#Exercises\_.28remote\_branching.29\_.E2.8C.98

Exercise 19 - killing one stone with many birds

19. Git rebase1(watch changes in 'gitk --all &'):

a) in 'master' add 3 revisions to 'logo\_jq.js'

b) create branch 'lackyLuke' and switch to it (git checkout -b):

- start it from HEAD~2

- add 3 revisions, every each to a different file (logo.html, logo1.html, logo2.html)

c) create branch 'sanchoPancha' and switch to it

- start it from HEAD~2

- add 2 revisions, all to the same file (logo.css), different lines

d) rebase branch 'lackyLuke' to 'sanchoPancha'

e) rebase branch 'master' to 'luckyLuke'

(rebase with conflict)

19.1. Git rebase2

a) create branch 'mniam' and switch to it

- git checkout -b mniam

b) add 3 revisions

- echo mniam >> logo.html

git commit -m "mniam" -a

- echo mniam1 >> logo1.html

git commit -m "mniam1" -a

- echo mniam2 >> logo2.html

git commit -m "mniam2" -a

c) watch changes in 'gitk'

- gitk --all &

d) go back to 'master' and make 2 revisions

- git checkout master

- echo mniam3 >> logo.html

git commit -m "mniam3" -a

- echo mniam4 >> logo1.html

git commit -m "mniam4" -a

e) refresh gitk (F5)

f) rebase branch 'mniam' to 'master'

- git branch

- git branch -a

- git checkout mniam

- git rebase master

- git status

g) solve the issue

19.2. Comparison of 'merge' and 'rebase'

a) in 'wc1'

- create branch 'mergy' and commit 3 revisions

- create branch 'merger' and commit 2 revisions

- combine those 2 branches with 'merge' (merger into mergy)

b) in 'wc2'

- do the same two first operations as in a) but use names 'reby' i 'rebaser'

- combine those 2 branches with 'rebase' (rebaser on top of reby)

c) use 'gitk --all &', to compare both ways

19.3. Reverting 'rebase'

(git reflog, git reset)

Exercise 20 - can git do necromancy?

20. Recreate file "logo5.html" in 'wc1'

(git revert?, git checkout?, git show?)

Exercise 21 - tagging

21. In one of your 'wc' create tags:

a) v1.0 and pin it to the revision, in which were made catalogs 'pic' and 'js'

b) v1.1 and pin it to the revision, in which was made catalog 'html'

c) v1.2 and pin it to the revision, in which was added file 'logo.css'

d) send tags to remote repo

(git tag)

Exercise 22 - why not use GUIs?

22. Use 'GUIs' to make/observe changes (gitk, gitgui, tig, IDE plugin):

a) add new file to repo

b) move existing file into different place

c) remove file from repo

d) make a conflict and solve it

e) push changes to remote repo

Exercise 23 - simple spoiling the config

23. Connect 'manually' catalog slideshow\_ with remote 'slideshow.git'

and get all freshest changes from repo (pull)

(git remote, git config -e)

Exercise 24 - the cherry on the cake

24. (git cherry-pick)

http://training-course-material.com/training/Git\_-\_Branching\_and\_Merging#Cherry\_Picking\_.E2.8C.98

Exercise 25 - more plumbing?

25. (git clean, git fsck, git gc, git prune)

25.1. In 'wc1':

- change logo.html

- add new file 'mniam'

- clean working copy

(git clean, git checkout)

25.2. In 'wc2':

- search for dangling objects (git fsck)

- look at the changes (git show)

25.3. In 'wc3':

- create 'detached head' state and make 2 revisions (checkout <rev>)

- go back to 'master' and search for 'dangling' objects (fsck)

- clean 'danglings' (gc)

- double check with 'fsck'

25.4. In 'wc1':

- repeat first 2 steps from 25.3.

- search for 'danglings' with 'git prune'

- clean 'danglings' with 'git prune'

Exercise 26 - can I be lazy bastard with git?

26. Create aliases:

a) g (git status, in linux)

b) git l (git log --pretty=oneline --abbrev-commit --abbrev=4 -25)

(use 'git config')

c) git save (git add, git commit)

(manually edit '.git/config', git config -e)

Exercise 27 - uglyfying history

27. Additional revision descriptions:

a) in 'wc1':

- to tagged revision add 2 additional comments

- show list of comments for that revision

(git notes)

Exercise 28 - externals

28. (git submodule)

http://training-course-material.com/training/Git\_-\_Submodules

28.1. (git subtree)

http://training-course-material.com/training/GIT\_-\_Patrick2#Submodules\_.28con.27t.29.E2.8C.98

## Setup the clean example

cd ~

mkdir s

cd ~/s

mkdir subtr

cd subtr

echo "subtr" > subtr

git init

git add .

git commit -m "subtr example"

cd ~/s

git clone --bare subtr subtr.git

cd ~/s/subtr

git remote add origin ~/s/subtr.git

git push --set-upstream origin master

cd ~/s

mkdir mainproj

cd mainproj

echo "mainproj" > mainproj

git init

git add .

git commit -m "mainproj"

cd ~/s

git clone --bare mainproj mainproj.git

cd ~/s/mainproj

git remote add origin ~/s/mainproj.git

git push --set-upstream origin master

## Include 'subtr' as remote repo and make it "subtree"

git remote add subtrOrig ~\s\subtr.git\

git fetch subtrOrig --no-tags

git checkout -b subtr subtrOrig/master

git checkout master

git read-tree --prefix=sub/ -u subtr

git status

git commit -am "tarball from subtr"

cd ~/s/subtr

echo "ch1 in subtr" >> .\subtr

git commit -am "ch1 in subtr"

git push

cd ~\s\mainproj\

echo "ch 1 in main" >> .\mainproj

git commit -am "ch1 in main"

git checkout subtr

git pull

## Merge changes from the subtree into main project

git checkout master

git diff-tree -p subtr

git merge --allow-unrelated-histories --squash -s recursive -X subtree=sub subtr

git diff-tree -p subtrOrig/master

Exercise 29 - slow motion (no pull..)

(clean 'wc1' and 'wc2')

29. Without 'pull':

a) in 'wc1' change file, send changes to remote repo

b) in 'wc2' get the changes from remote repo, but without using 'pull'

- go to '.git/' and look at the content of file 'FETCH\_HEAD'

c) combine changes with main branch and send it to remote repo

(git fetch, git merge)

Exercise 30 - no secrets in git objects

30. Searching for objects in repo:

a) in 'wc3' search for all files, which have linked 'js' files, so they will

have rows with both '<script' and '</script>' html tags

(git grep)

Exercise 31 - life after the death of wikipedia

31. Don't use uncle gooogle and find out how to disable possibility to force 'push'

(git help)

31.1. Same way, add another remote repo in your 'wc1'

(git help remote)

Exercise 32 - automatic bug-finder

32. (git bisect)

http://training-course-material.com/training/Git\_-\_Bisect

Exercise 33 - whatever

33. Ignoring files (.gitignore vs .git / info / exclude)

a) in wc2 edit '.png' file in 'Inkscape' and save it as 'test.svg':

- add svg to the tracked list of ignored files (.gitignore)

- send changes to the repo

b) in wc1 edit the logo1.html file in the gedit program and change all 'blindZ' to 'up'

- do not save changes yet

- make sure 'gedit' will automatically backup the file

- add a copy to the non-trace list of ignored files (.git / info / exclude)

- send changes to the repo

c) clone repo to the next copy of wc3 and compare both ignore paths:

- edit '.png' file in 'Inkscape' and save it as 'source.svg'

- edit the 'logo2.html' file in the gedit program and change all rotations to 'down', wait for a copy

- check the state of wc3, what can you say?

- send changes to the repo.

Exercise 34 - listing files with context

34. List files in 'wc2', which are:

- deleted

- on the stage

- ignored

- with additional info about changes in cache

(git ls-files)

Exercise 35 - being a good pimp with git

35. Hooks.

a) in 'wc1':

Create 'pre-hook', which will check if in staged changes in files

is word 'notFixedYet' and it will not allow to commit such change:

- copy 'pre-commit.sample' into 'pre-commit' in folder '.git/hooks'

- make sure it is executable

- prepare proper bash code

- change 'logo.css' with "/\* notFixedYet \*/"

- change 'logo.html' and type "Fixed"

- try to commit with 'git commit -a' (?)

- use simple commit

- any conclusions?

- send changes to remote repo

b) in 'wc2':

- actualize repo

- add "/\* notFixedYet \*/" in 'logo.js'

- commit changes

- what can you say?

- fix it! (-:

35.1. Post-hook

a) in 'wc3':

Create 'post-hook', which will ask db (mysql) with simple query (select \* from table\_name),

but only when some one will switch between branches

Exercise 36 - continuous integration

36. CI basics

a) Create free account here https://about.gitlab.com/pricing/#gitlab-com

b) Add new project 'slideshow\_'

c) Clone it into local repo (git clone <url>)

- add all files and do initial commit (git add ., git commit -m "...")

- push (git push -u origin)

d) Make new issue from 'Exercise 1', point a)

- solve it, commit and push

- pull it in your local copy

e) Create new issue from 'Exercise 17'

- try to solve everything in gitlab (no command line at all)