VERY BASICS OF GIT BRANCHING - OUR SIMPLE TWO-LEVEL updated 2021-09-30

==

0. first naturally **git clone https://full-link-to-the-repo.git** in some folder for your repos, e.g. c:\git_repos\SWP\
    - will create a local copy of the repo as same name folder to the folder where you are
        i. thus remember to run **pwd** and **ls -Falls** commands a lot to be folder-aware
        ii. need to do **cd repo-folder-name** to get to the folder. (Use often: **pwd** or **ls -Falls**)
    - sometimes projects have long folder paths, thus put your repo folders either to c:\users\joe\git_repos\ or even c:\git_repos\ and not under longer paths
    - when ls -Falls shows the **.git** folder, you are in the repo root folder
    - I used root root folder SWP so that when I open it in e.g. VS Code I see the few related repos in editor at once, and the git integration still works!

<u>in branch main, local computer</u>

git pull - Get latest of main branch for reading

- you have not edited anything yet => straight forward, you get latest version of everything without merge = you get the latest commits from remote = GitHub.com or similar.

(read the current version well)

git pull   - Again to see if anything changed while reading/studying the code

- (plan the changes now)

git pull   - When ready to do the changes, again to see if anything changed while planning...

git checkout -b product-getAll-filtering-add-juhani

- clones the local main to be a new branch of yours, plus moves to that new dev branch!
- look at the branch naming. It’s written backwards so that people get the most important info first, e.g. this branch would be about ’products’. People working with other things know if it’s relevant to them right now or not.
- Also the developer name(s) at the end is useful as in many cases we just see the branch name, not contributor(s)

<u>in dev branch, local computer</u>

git push --set-upstream origin product-getAll-filtering-add-juhani

- this will create a copy of the local dev branch to the remote = origin = GitHub.com/Bitbucket/or so
- it’s important to show others in origin=remote=GitHub.com that you have started changes on something
- here you see why branch name matters!

(as fast as possible, do the changes in local branch, do not leave branch open for more than an hour or two)

git add -A

- adds All the changes to local files to the ’staging’. (Thus creating kind of a candidate for commit)

git commit -m "Add filtering to GET all products API end-point"   (commit message)

- now the changes have gone to version history!
- for local branch only!
- for the branch that was active at that time!
- and the staging area is cleared!

git push

- now the commit will be delivered to remote copy of the dev branch too.
- There should not be any newer commits in remote as this is your dev branch. Should be straight forward
- (No need for full command anymore: git push --set-upstream origin product-getAll-filtering-add-juhani)

in GitHub.com or BitBucket or such web page interface

(Go to GitHub and create a ***Pull request*** - Request your branch changes to be reviewed + merged to main branch)

(Ask someone to review your pull request and accept or reject it)

(Someone will accept and merge your changes = the pull request)

(Delete that dev branch as obsolete in remote - Indicates also to others you are not working on it anymore)

- either delete the branch in GitHub web interface or
- **git push origin --delete product-getAll-filtering-add-juhani** from the command prompt

in Local pc

git checkout main

- go back to main branch (earlier called master)
- the command will say that your branch is uptodate with origin main **which is not true!**

git pull (Do this now so you won't forget to do it, your main does NOT have latest from remote main)

git branch

- list of branches in your local PC
- active branch with * mark

git branch -D product-getAll-filtering-add-juhani

- deletes the dev branch from local repo
- if need the branch later again, you could simply just create it with the same name!
- so absolutely no use to keep the branch existing when **not making changes**

Then start again from the beginning with…

in Local pc, branch main

git pull

You can also learn to use **git status**, **git branch** and **git pull** on regular basis.

Principles
all in all, having own branches open for _shortest possible time_ is good in so many ways:

- less merge conflicts
- common parts shared earlier to other developers
- knowledge shared sooner to other developers
- it develops the refactoring and SRP/DRY/architecture thinking

Trying out something risky?
What if you want to have a tryout branch for checking some solution? Mostly same initial process, just (no obligatory remote branch copy), no pull request and no merging to main. Maybe add word tryout to the beginning of the branch name?

WHAT ARE THE ALTERNATIVES TO OUR SIMPLE MODEL?

==

Advanced gitters option
If you are fluent with merging tools, you can also, while working in your branch, merge the changes from remote main branch to your working branch: (Do this only if you still publish your own work to remote main very fast!)

git pull origin main (Then your branch, if merge succesful, would have less changes compared to main, when later make the pull request)

* * *

- THREE-LEVEL BRANCHING Production = running version, Development/Test = common test version, Feature = developer's own additions to be merged to the Development and tested there.

(But we do not need production version, we do not have customer using the product right now, thus kind of the Production level is not needed, 2-levels enough in our simplistic model)

- PULLING FROM COMMON BRANCH CHANGES TO YOUR FEATURE/ADDITION BRANCH

Good: Your branch will not deviate too far from the common

Bad: Other people will not get your work soon. In our project e.g. frontend team needs your backend service fast (now that we do separated way, later we of course can do full-stack feature teams)

**) Hybrid: You can do mostly the simple way above, changes published fast to remote main branch. But from time to time, when seems appropriate, pull from main/common/master to stay up-to-date with the changes there. If fast to merge to you work.