PB173

Perl

Table of Contents

Setup

This document explains how to use PB173 Perl Git repository and how to submit homeworks.

Create a project

1. Navigate to https://gitlab.fi.muni.cz/xlacko1/pb173-perl and fork the repository.
   Make sure you set **Visibility** to **Private**.
2. This next step is **extremely** important, do not skip it!
   Remove the fork relationship:
   Settings → **General** → **Advanced** → **Remove fork relationship**.
   The rest of this manual should still apply, except the step 3 below.
3. Give access right to your tutor.
   Settings → **Project information** → **Members**.
   Here, invite your tutor (`xlacko1`) with role Reporter.

Create a SSH key and add it to your account in GitLab to make authentication easier.

Local repository

Clone your repository on the computer you wish to use when working on tutorials or homework assignments. This can be Aisa, your own personal computer, someone else's personal computer you hacked into, ...

1. **Clone the repository.**

   In the project dashboard, copy the URL (SSH if you have set up your SSH keys, HTTPS otherwise) and clone it on your computer, for example:

   ```
   $ git clone git@gitlab.fi.muni.cz:‹LOGIN›/pb173-perl
   $ cd pb173-perl
   ```

2. **Set up upstream repository.**

   From now on, we will call your repository "*origin*", and the official repository belonging to the tutor shall be known as the "*upstream*".

   Create a new remote in your repository called *upstream* to point to the official repository. Use either SSH or HTTPS link:

   ```
   # HTTPS
   $ git remote add upstream https://gitlab.fi.muni.cz/xlacko1/pb173-perl

   # SSH
   $ git remote add upstream git@gitlab.fi.muni.cz:xlacko1/pb173-perl.git
   ```

Updating materials

Initially, branch `main` contains a single (empty) initial commit. This branch is for you to do as you please, and where you are advised to work on your tutorials. You can create other branches if you wish.

Before every tutorial, new materials will be pushed into the `seminar` branch of the upstream repository. All you need to do is merge them into your `main` branch.

Make sure you start in your `main` branch:

```
$ git status
```

Now update the materials:

```
$ git fetch upstream seminar
$ git merge upstream/seminar
$ git push
```

When you are finished with the tutorials, you are advised to commit and push what you have done to your origin repository.

Corrections

If an error is found in tests or source materials, the correction will usually appear in the `seminar` branch, and the above commands will be enough to fix it.

However, if an error is found in a homework after you are expected to have modified it (and you did), you will be advised to update with some named commits using cherry-picking.

⚠ If you have **not** yet started with the affected homework, just update your materials!

```
# MAKE SURE YOUR REPOSITORY IS CLEAN
# TIP: See ‹git stash› below

# Update main branch as above
$ git switch main
$ git fetch upstream seminar
$ git merge upstream/seminar
$ git push

# Get back to your homework
$ git switch HOMEWORK-BRANCH
$ git cherry-pick COMMITS…      # COMMITS… will be specified in the forum
```

If this results in conflicts, either you can try to fix them manually or reach out to your tutor.

Homework

Each seminar will contain a few homework assignments, or tasks. You can choose any subset of these tasks to work on; even empty.

Since this seminar is quite small, we will not use any additional tools, GitLab will be enough. The solution will therefore be submitted as Merge request (MR from now on).

If you decide to solve more than one task, you have to submit each of them in a separate MR.

1. **Start from a clean repository.**

   Use `git status` to confirm you are in your main branch with no changed files.

2. **Create a branch for your homework.**

   The name of the branch must have the homework code including the last letter, for example `hw03b`. Then push your new branch to GitLab:

   ```
   $ git switch --create hw03b
   $ git push --set-upstream origin hw03b
   ```

   If `git switch` is not available (usually on older systems), use `git checkout -b hw03b` instead.

> *Old Git on Aisa*
>
> Aisa's distribution is old and does not understand `git switch`. There, you have the following options:
>
> 1. Use `git checkout [-b] BRANCH` instead of `git switch [-c] BRANCH`.
> 2. Use Git from modules, `module add git-2.25.1`

3. **Work on your homework.**

   In the homework branch you should only modify files regarding the task. Create commits regularly, especially when you want to switch to a different branch. Don't forget to use `git push` and `git pull` as well if you want to access your work from different computers.

4. **Create a MR and submit your homework for review.**

   When finished, make sure you made `git push`. Then create a MR:
   Project's left panel → *Merge requests* → *New merge request*

   - In *Source branch*, select the branch with your homework.
   - Make sure that *Target branch* names your project's main branch.
   - Click *Compare branches and continue*.

In the main MR form, fill in a few details:

- The *Title* **must** begin with `Draft: CODE` where `CODE` is the homework code, e.g. `hw03b`. You can add `:` and your own title after this.
- Add `xlacko1` to *Reviewers* field.
- (Optionally) set *Assignee* to yourself.
- (Optionally) check *Delete source branch when merge request is accepted* option.

You can set other options (like *Description* or *Labels*) as you please.

Finally, click *Create merge request*.

5. **Now you wait.**

   Do **not** Merge your homework immediately. This is where the `Draft:` part in the title comes in handy, as GitLab will prevent you from clicking *Merge* as long as this prefix remains there.

6. **Review.**

   You will likely recieve review for your solution in GitLab as well. In rare cases the review might be delivered by an e-mail or an owl.

   If your solution is good enough and clean, your Merge Request will be approved. Otherwise you are expected to work on the task a bit more. Read the review carefully.

   There is no need to create MR for the resubmit, simply switch back to your homework branch, make changes as necessary, and then commit and push them. GitLab will automatically update the MR.

   Avoid squashing commits or using `git push --force` for review. While these techniques are useful in production environments, they can make homework reviews and updates quite messy.

7. **Acceptance.**

   Once your homework is accepted, you will receive a mark in IS for each accepted task. Whether you merge this branch into your `main` repository is up to you.

Tips and tricks

1. Use `git status` before each `commit` and `push`.

   This will help you to avoid messing up branches or creating merge conflicts.

2. Do not leave uncommited work.

   Always create commits for something you have done or finished, especially if you wish to switch branches.

3. Stash.

   If you want to switch branch, or do something that requires a clean repository, but you have unfinished work that is not worthy of a commit yet, look at the `git stash` command (`man git stash`).

   Short version:

   ```
   $ git stash
   ```

   Take uncommited changes and store them in a stage area (you can imagine it as a virtual commit) without making changes to your current branch. You will end up with a clean repository.

   ```
   $ git stash list
   ```

   List stashes, useful when you have more than one.

   ```
   $ git stash pop [stash]
   ```

   When you want to resume working on the stash, use this command to restore changes on top of your **current** branch. You may wish to specify the stash if you have more than one.

   You can use `git stash` before `git merge` or `git pull` to temporarily clean out the repository.

4. Graph.

   While there are some external tools to visualise commits and branches of your repository, Git can do that too:

   ```
   $ git log --oneline --decorate --graph --all
   ```

   You can create a handy alias for this command, like

   ```
   $ git config [--global] alias.graph 'log --oneline --decorate --grah --all'
   $ git graph
   ```

Do not be afraid to ask for help

If you get stuck with Git or you think you messed up, *do not panic*. Git is a very powerful versioning system, and while it tries to protect you from trouble, it is not perfect. Things like these do sometimes happen.

Do not hesitate to ask for help in IS MU Discussions or on Discord.