Course
ELEC-A7151
Course materials
Your points
Code
Code Vault
Course Pages
MyCourses
Teams Channel

« 2 Software libraries Course materials 4 Valgrind »
ELEC-A7151 / Software development tools / 3 Git

3 Git¶

Contents

3.1 General overview¶

In any larger software projects with multiple developers who work on the same code, **version control systems** are commonly used to synchronise the work between participants and track the changes in better way. One important function of version control systems is to handle parallel editing of the source files to avoid conflicts if multiple project participants edit the same locations in source files. Also single-person coding benefits from version controls systems for example, when working on multiple computers.

In version control systems the codebase is stored in **repositories**. The *repository* holds all the files and their current state, with the change history. With version control systems all new code, be it a new feature, or a fix to an earlier feature, is separated into logical **commits** that are stored into the repository. A *commit* can be reviewed and compared by each project participant, and each commit is logged with its contributor's name and timestamp, along with a short text description about what was changed.

Git is currently one of the most common version control systems, which is also used on this course. For the purpose of this course, the exercises use a private repository where only you will do modifications, whereas the the project uses a private repository where you and your team mates make changes.

In a typical setup, where multiple developers work with the same project, there is a master repository on a server that is accessible to all developers. Each developer **clones** a copy of the master repository to their own workstation that can be worked on. After making modifications, i.e. new commits, to their local repository, the developer pushes the changes back to the master repository so that they will be available to others. If there have been other changes in the meantime, Git merges the changes into a single common version of code. Occasionally, a developer also wants to **pull** any possible new modifications from the server into their own local version, to stay up to date of the modifications others might have done. This workflow also applied to a single developers working on multiple computers.

3.2 Setting up¶

Git is often used through a command line interface, which we also assume in the below examples, although most Integrated Development Environments (IDEs), such as VSCode, have git integrated to the graphical system and its menus. If you're on Windows, we recommend that you install WSL according to the instructions below and use that with Visual Studio Code for git access.

3.2.1 Enabling access to Git server¶

HTTPS vs. SSH¶

You can use `SSH keys` or `https` to access your remote git repository. When using `https`, you'll need to login with your username and your password when connecting to the remote repository. SSH keys automate this process.

Adding SSH Keys¶

From the top right corner you can access settings, and from there you can add an SSH key to your account. To see instructions on how to generate an SSH key with your operating system, see Github instructions, section Generating a new SSH key. If you're in WSL, check the Linux instructions! After generating the key, add it to your account, see picture below. You can print the public SSH key to the terminal with `cat ~/.ssh/id_rsa.pub`.

After adding the key to your account, you can access your repositories easily. You'll need to repeat the process for each computer you'll want to use your repository from.

3.2.2 Cloning a repository¶

Now you can go back to the front page (Dashboard). Open your personal repository, from the links on the right side. From the repository's site, you can see the URL which will be used to clone the repository to your computer.

From the git command line, you'll clone the repository to your preferred location. Navigate to the location that you want to clone the project to with the command `cd`.

After that, run the following command if you prefer to clone using HTTPS:

```
git clone https://version.aalto.fi/gitlab/cpp-autumn-2021/<topic>/<topic>-group-<id>
```

for SSH:

```
git clone git@version.aalto.fi/gitlab/cpp-autumn-2021/<topic>/<topic>-group-<id>.git <folder-name>
```

This clones the remote repository into a folder called `<folder-name>` in the location you were at. You can choose another name for the folder if you want, or you can omit it. If you omit the `<folder-name>`, git client will create a folder `<topic>-group-<id>` in your current folder, and clone the remote repository in it.

3.3 Recommended workflow¶

1. First pull from the remote to get the latest changes (git pull).
2. You can now modify the code.
3. Test your development locally.
4. When you are ready with the modification, make a commit and push it to the server (git push).

3.3.1 Getting the latest changes¶

You can update your local repository with the new remote versions with:

```
git pull
```

If everything goes well, you'll either see `Already up to date.` which means that there are no updates available at this time or

```
> Updating xxxxxx..xxxxxx
> Fast-forward
>  somefile.cpp                |    xx ++++-
>  xx file(s) changed, xx insertions(+), xx deletions(-)
```

which means that the update was completed successfully with no problems.

Merge conflicts when pulling¶

`git pull` command might introduce some **merge conflicts**. You are informed about them somewhat like this:

```
CONFLICT (content): Merge conflict in somefile.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

These need to resolved by you. If you know you haven't changed the file listed in the merge conflict, you can just checkout the remote version by doing `git checkout --theirs somefile.cpp` and then `git add somefile.cpp`. That will resolve the conflict. If you have changed the file yourself, you'll have to open it to merge it by hand. When you open the file, you'll see something like this:

```
<<<<<<< HEAD
your code and additions here
=======
the course personnel's changes here
>>>>>>> upstream/master
```

What you'll need to do, is to remove the `<<<<<<< HEAD` and `>>>>>>> upstream/master` tags, the divider `=======` and figure out how you'll want the code to look like with the changes "merged". After you've fixed the conflict, you'll need to add the file in git to mark the conflict resolved `git add somefile.cpp`. Running `git status` will show you if all the conflicts have been resolved.

When you've sorted out all the conflicts, you'll need to create a commit that concludes the merge. Running `git commit -m "Merge upstream"` and `git push` should get the merge concluded and you can continue your work as usual.

3.3.2 Committing and pushing to the server¶

When you have reached to a satisfactory point, you might want to (or should) create a development log to show your collaborators what you have done by committing your changes. When you want to send your local commits to the Git server you can push them to the remote server.

You can use either the command line or Visual Studio Code integrated git.

Visual Studio Code

Committing and pushing in fairly simple. See this video for example:

When pushing for the first time, you'll encounter a request to add your email and name to the git config. Here's a video how:

Command Line

From the command line, you can see the files you've changed with `git status`.

```
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
    (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working directory)

        modified: first.cpp

no changes added to commit (use "git add" and/or "git commit -a")
```

To create a commit that saves these changes to git, you'll need the command `git commit`. So, run the following when you want to save your changes to git:

```
git commit -am "Commit message here"
```

`git commit` command with the parameters `-am` does the following: `a` adds all the changed files to the commit, and `m` adds a message which is visible in git.

Now, if you check your `git status` again, you'll see you have a commit that can be pushed.

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

You can push it with the command `git push`.

3.4 Projects and git¶

In projects, where multiple developers work with the same project, there are some features in git and the web interface (gitlab/gitea/github) that are very useful for organizing work between the developers.

3.4.1 Issues¶

Issues can be used to track tasks and features for your project and assign them to specific people. They can be grouped based on labels (like UI, core logic, extra feature etc), so it's easy to see which part of the project still has work left. GitLab has a more detailed introduction to issues, which should work the same way also in `version.aalto.fi`.

3.4.2 Branches¶

Branches are used to develop features separately from the main branch. As a good rule of thumb, the main branch should always contain a working version of the project, whereas the feature branches can have some incompatible work and issues that are still worked on. To start a new branch from the current status of the project, use command `git checkout -b branch-name`, which will create a new branch with the name you supplied and take all your local changes there. After that you can continue working as normal, developing the feature in your own branch. You can still push changes to the remote repository with `git push`.

To learn more about branching, you can use this website to practice.

If your development is taking long and others have added new content to the main branch of the repository, it is good to occasionally merge their changes to your branch. You can do that by pulling the latest changes with `git pull` and then on your feature branch: `git merge master`. Fix any merge conflicts you might get, and then your branch is up-to-date with the main branch again. When you've completed the feature you've been working on, a good practice then is to make a pull request.

3.4.3 Pull requests¶

Pull requests are a way of merging new content to the main branch of the repository. By opening a pull request you can create an overview of your changes. A good practice is writing a short overview of your changes to the message, which helps others to understand what the code does. After you've opened a pull request, others can review your code and leave comments or questions. Then you can still make modifications to your branch if there are some improvement suggestions and after that you can merge your changes to the main branch. Pull requests also will show if there are any conflicts you need to resolve before merging your changes.

3.5 List of Git commands and other useful links¶

- Most common git commands
- 15 minute GitHub code school
- Aalto's git instructions
- Git cheat sheet

« 2 Software libraries Course materials 4 Valgrind »