Introduction to Git

and GitLab

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Git in the course

- In this course we will be using *git*, a distributed version control system.
- The main purpose is to store, track changes and share source code.
- Git is today an industry standard that is wildly used for version control.
- Several of you might already use it, but we will also introduce our own instance of GitLab in this lecture.
  - Otherwise the most common name among git is GitHub.
- This lecture will give you some background, some rules for using our GitLab and some guidelines for your workflow.

Background

- Version control is something that is needed when a project gets larger.
- On top of that, it also backups your files, which why it could be used even for smaller projects.
  - Manual version control could be to backup your files to folders on a cloud drive.
- During the years, several *Version Control Systems* or VCS have been developed and used.
  - CVS, Subversion and Visual SourceSafe to name a few.
- Many, if not most, so far have been *centralised*, which means that they have a server somewhere holding all the files.
- Git, any many like it, is *distributed*, meaning that everything (yes, everything) is stored locally.
  - But with the option to send it to a server as well for safe keeping.

Short, short history of Git

- Git is the brainchild of Linus Torvalds, developer of the Linux kernel.
  - Git means *unpleasant person* in British, which is something Linus sometimes identify with...
  - It could also mean “global information tracker”...
- It was developed in 2005 after the previously used DVCS BitKeeper revoked Linux’ free-of-charge license.
- It has evolved significantly since 2005, but the keywords *speed*, *simple design* and *fully distributed* still remain.
- Today it is more or less the defacto standard when using versioning control, both for open source software and commercial.

GitLab, GitHub and using git in class

- Git works as a “time machine” on your local machine, making it possible to keep track of changes (and revert if necessary).
- However, often we like to also make a backup somewhere and this is where sites like GitHub and GitLab come in.
- GitHub is perhaps the most wildly used and know hosting company for git.
- At LNU we have opted for setting up our own instance of *GitLab* instead. `https://gitlab.lnu.se/`
- GitLab is an open source DevOps lifecycle tool that began its life in 2015.
- GitLab is the platform we use to have a centralised storage for your files.

On coming lectures

- This first lecture was just an overview and background.
- Several other videos are available and while they are meant to be viewed back to back, they work in isolation as well.
  - Especially if you have prior knowledge of git and working with it.
- These videos will cover:
  - Setting up git and GitLab
  - Working with git in isolation
  - Working with git from two or more computers
  - Using git for group work
- It is important to get an understanding of git and GitLab as it will be used wildly in future courses.

Setting up git

and GitLab

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Getting Git

- To be able to use git, you need to download the software.
- The main web page for git is `https://git-scm.com/`
- Follow the links to download for the appropriate platform.
  - Linux and Mac users can also download from respective repositories.
- Windows users: make *sure* you install *Git Bash* (an option when installing).
  - Also make sure that you install *OpenSSH* and use the standard configuration.
- We will primarily use git from the from the command prompt, therefore it is important to install Git Bash for Windows users.

Working with Git

- In Linux and macOS, use the ordinary terminal software – in Windows open the start menu and search for Git Bash.
  - This course is not about using the command prompt, if you are unfamiliar with it use the Internet.
  - A nice starting point could be: `https://towardsdatascience.com/basics-of-bash-for-beginners-92e53a4c117a`
- Navigate to a folder (directory) where you have or will have your files for your course.
- Notice that we will try to use the terminal as much as possible, but much of what we show can be done in other ways too.
- You will *have to* set a name and an e-mail, and we recommend that you set up *gitignore* as well.

GitLab at LNU

- Log in on `https://gitlab.lnu.se` with your student account.
- Have a look at the Wiki which deals with most of the things you need to know.
- To get started, look at the documentation at `https://gitlab.lnu.se/instructions/students/get-started`
- For each course you take, you will get a number of repositories.
  - This will be predefined for you, however, if the don't show up – contact us.
- When first arriving to GitLab, you need to do some setting up.

Setting up GitLab

- GitLab will be using *ssh keys* rather than username/password for authentication.
  - This is a lot safer and will make pushing easier as you will not need to type username or password.
  - You will have to create private and public keys and store the public on GitLab.
- The following image will be visible on each login until you have set up ssh correctly:

- Follow the instructions you get when you press it to add your ssh key to GitLab.
  - If you already have one, it is located in *.ssh* in your home folder.
  - If you don't, the page will show you how to generate one.

Working with git

using GitLab

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

A new project

- You will either be greeted with an empty project folder or with something pre-defined by us.
- If the folder has files, clone them to your directory by pressing the button and copy the code to the terminal.
  - Remember to place yourself where you want the project to be cloned.
  - Issue something like:
    `git clone git@gitlab.lnu.se:1dt901/student/tanstudent/assignment-2.9.git`
- If the project folder is empty, there are instructions there to initiate a new project locally.
  - Use the terminal to navigate to the correct place, and issue the commands on the web page.
  - Look for the commands for “Push an existing folder”

Push an existing folder

- Follow these instructions to initiate your local repository if you *do not* clone:

```
cd existing_folder
git init
git remote add origin git@gitlab.lnu.se:1dv000/student/xx222xx/assignment-1.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

- We leave them here as a reference for future use (and the exact course code and account name will be yours).
- These instructions will now be explained a bit further.
  - Except for the once about navigating in the terminal...

Basic workflow

- To initiate the local git repository, go to the folder where you have your project and in the terminal and execute:

```
git init
```

- This will create a number of hidden files.
- To add a file to ignore your binary files and IDE (like VS Code, PyCharm and so on) specific files, use:

```
git ignore python,visualstudiocode >> .gitignore
```

- **IMPORTANT:** also add your operating system to the above (linux, macos or windows).
- To get the current status of git use:

```
git status
```

Important concepts

- From an overview, four important concepts of git are:
  - **Working directory** where you have all your files (souce code).
  - **Staging area** where updated files are marked to be committed, but are not yet committed.
  - **Local repo** holds all committed files (over time).
  - **Remote repo** stores your files on a remote server, with history.
- A *repo* is a repository of your files.
- *Committing* a file means that it is version controlled in your repository.
  - You can go back to a previous version if you need to.
- The *remote repo* is our instance of GitLab.

More basic workflow

- To add files to the stage area:

```
git add [filenames]
```

- To commit the file to your repository use:

```
git commit -m "Commit message"
```

- A `git status` should now say that everything is checked in.
- Next step is to push it the remote repo, that is to `https://gitlab.lnu.se/`
  - See previous slide for adding a remote repo.
- Push, that is send the code to the server, use:

```
git push -u origin master
```

Working with Git

- Make frequent commits to your repo.
- It is not okay to make one large commit just before the deadline.
  - We will not impose a number of commits, but several are needed with some time between them.
- Make a habit of commiting (and pushing):
  - when you take a break
  - at the end of the day
  - when a task is done
- Failing to commit and push is not reason for failing the assignment alone, but will not be in your favour.

Working with assignments

using GitLab

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Handing in assignments

- Assignments are handed in as *merge requests*.
- More information on `https://gitlab.lnu.se/instructions/students/get-started/-/blob/master/submit-an-assignment-as-a-merge-request.md`
- If your assignment has *issues* they should be closed before handing in.
  - Issues are requirements that your teacher has created for you.
- When you feel *done* with an assignment, create a new merge request.
- For source branch, select *Master*
- For target branch, select *Release*

Setting up the merge request

- Title should be the assignment name.
- Under *Description* you must pick the *Release* template.
- In the release template, tick the boxes.
- You may, but you don't have to, give further feedback to the teacher in the *Assignment Report* box.
- *Milestone* is for what assignment you are handing in (with a deadline)
- *Labels* should be selected for your programme and your language.
  - For example NGDPV and English.
- When done, press *Submit merge request*

Making updates to your assignment

- There might be automatic tests performed on your assignment.
- If these fail (you will be notified about that), you can correct your code *up till the deadline*.
- Do this by changing the code locally and commit and push it to the server.
  - Each time you make a push to server (and you have an ongoing merge request), the test will be performed again.
- After deadline, the teacher will look at the assignment and give you feedback.
- You know that the assignment is approved when the merge request is accepted and closed by the teacher.

Using Git with multiple computers

in different places

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Working with several computers

- Git also makes it easier to work with several computers.
  - Either as different individuals, or for your own sake to use different computers.
  - This video is about being one person accessing the code from different computers.
- If your changes are saved on GitLab from one computer, first use the *clone* command to copy it to another computer.
  - Using `git clone` on the command prompt as described before.
- With that, you have the code *in two different* places.

Staying up to date

- As the code is on two computers, it is important to stay up to date.
- Therefore, begin each coding session by updating your local copy with whatever you have on the server.
  - This is done by issuing a `git pull` on your computer (in the correct directory).
- Continue working with committing and pushing your code.
- When changing computer, do not forget to do a pull for the updated code.
  - If you forget, you might get a conflict when you push the new code.
- Git is a great tool for shifting computers, but you need to be structured and remember the steps.

Project work with Git

using GitLab

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Project work

- Git was created to make it (fairly) easy for many people to work on one project.
- To assist this, it is possible to create different *branches* of the code.
  - A branch is essentially a pointer to commits done in a specific order.
  - *Master* is the default branch, but there is nothing special about it.
- Branches can be created by the command `git branch [name]`.
- Notice that it does not move you to the new branch, only creates it.
- To move to the branch, issue `git checkout [name]`

Working on the branch

- It is a good idea to create new branches for each new feature to add.
- If possible, try to not overlap the work of others.
  - This could lead to conflicts – this will be covered later in this video.
- When adding and committing, this is now done towards the new branch.
- To push, you need to add a new upstream branch.
  - The first time, later it will be default.
- Do something like `git push --set-upstream origin testing`
  - Where `testing` is the name of the branch.

Merging

- From time to time, you need to merge your work with *master*.
- This is easist done via the web interface.
  - It is, of course, possible to do via the command line as well, but...
- On the page for the project select *Merge request* and create a new merge request.
- Select the branch that you have been working on as source and *master* as target.
- In the interface add any additional text you like, you don't have to do anything here right now.
- Press *Merge* and if there are no conflicts, the files will be merged.

Conflicts

- If, however, there is a conflict you will have to resolve this.
- On the web page, you can press a button for resolving conflicts.
- A *diff* between the files is shown and you have to select:
  - use ours or,
  - use theirs
- depending on which solution you like to have saved.
- There always need to be a human to select between these kinds of confllicts.

More on working with the branches

- When done merging, get back to what ever branch you need to be on.
  - This could be *master* or another branch for a new feature.
- Remember, however, to always pull the latest changes from *master* to be in sync!
- Delete branches you do not need with `git branch -d [name]`
- List branches with `git branch`
- Also use `git status` to get an overview of where you are and if you are up to date.

Strategies

- There are many strategies for how to work with branches.
- What you want to achieve is few conflicts and easy merging.
- One strategy is to have a *devel* branch just below *master* to which other branches are merged before going to *master*.
- It could also be a good idea to have one single person appointed *merge master* how is the only one allowed to merge with *master*.
- There are many other strategies available when searching the web – but begin by keeping it simple.