APPS@UCU

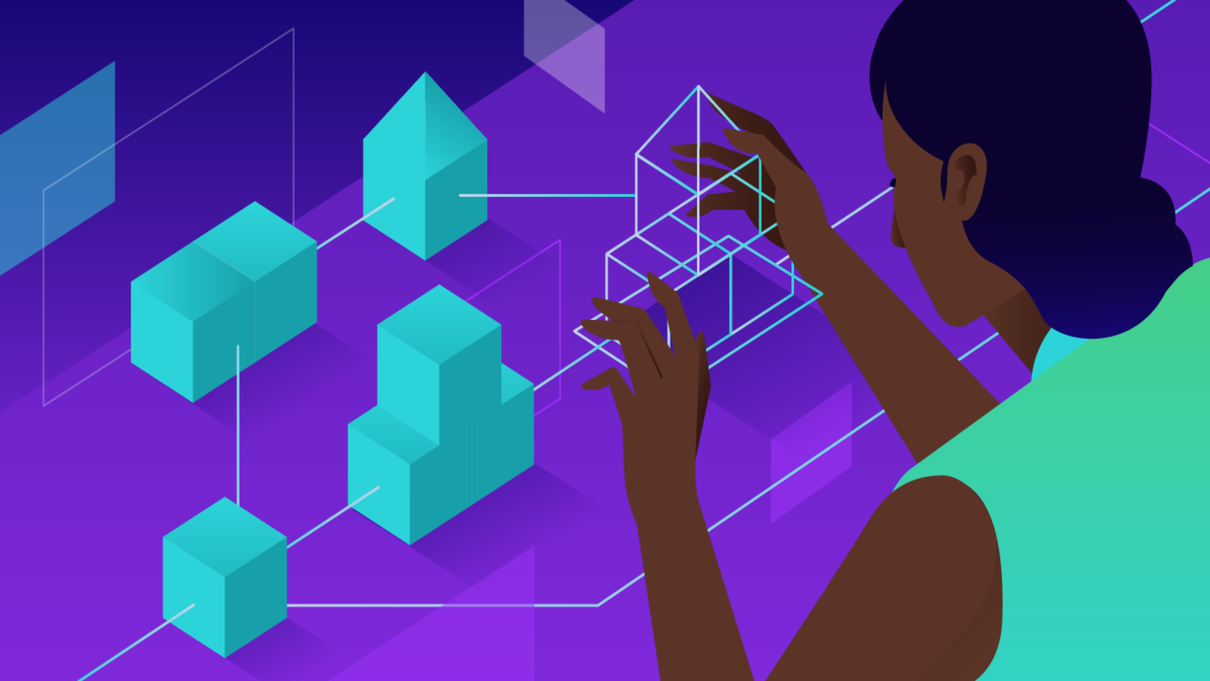
Linux course

Version control systems

Morhunenko Mykola

APPLIED SCIENCES FACULTY

Contents



Version control systems

- Sooner or later, during the development process, it is necessary to check, what was before, how it became broken
- Maybe it’s easier to use Ctrl+z , but it’s impossible to check what was three weeks ago with any keybinding
- So in 1972 people started to think about version control systems
- Firstly, it had been just a tool for saving a history of binary files, but in 1977 the first source code control system was introduced
- The main idea behind - to save the program source code on some checkpoints (commits) , add features, develop them leaving trunk untouchable (branches) , merge new features with a trunk, and release some tags
- Since than, the concept itself was developed, and a lot of version control sys-tems have apeared

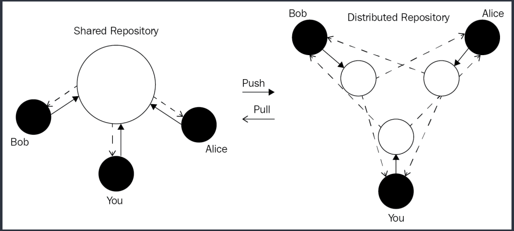
git

Linus again...

- Linux kernel is a huge project, and it is important to have some source-control management system (SCMs) to maintain it
- From 2002 to 2005 BitKeeper, a proprietary SCMs was used to maintain the project
- At some point (3 April 2005), Linus Torvalds realized, that existing tools are not suitable for Linux development, so in three days he anounced a project and become a self-hosting of Git on the next day
- It was totally different SCMs. Linus maintained it for half a year, and Junio Hamano has been the core maintainer since then
- It was open-source, free software, with a very strong safeguards against corruption, either accidental or malicious
- Torvalds sarcastically quipped about the name Git , means unpleasant person in British English
- He said: *"I’m an egotistical bastard, and I name all my projects after myself. First ’Linux’, now ’git’."* =)

What makes Git so good

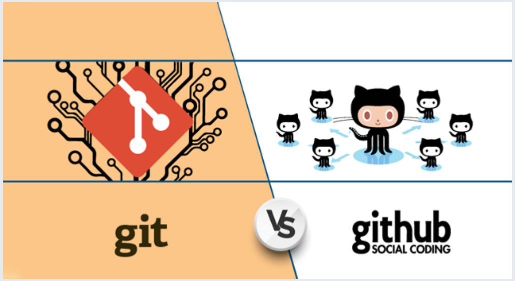
- Strong support for non-linear development
- Distributed development
- Efficient handling of large projects
- Toolkit-based design
- Pluggable merge strategies
- And more other features
- It’s hard to find any statistics, but that is clear - Git is the most popular SCMs of ourdays



github
SOCIAL CODING

Git is not GitHub

- GitHub, Inc - provifer of internet hosting for software development and version control using git
- It offers all the functionality of Git + it‘s own features
- Since 2018 - subsidiary of Microsoft
- Not an Open Source project , but there is a forum for feature requests...
- As of January 2020, GitHub reports having over 40 million users
- More about it’s features after Git usage part



```
$ git push
```

Git is not GitHub

- First difference - GitLab was created by Ukrainian people, in Ukraine =)
- It has deffinitely more features, than GitHub , but there is no critical difference
- It is Open Source , unlike github
- It is possible to have the same repositoru on both servers, and I do it sometimes
- So as for 2021 it's just a question of taste

GIT USAGE

IN CASE OF FIRE

Git Commit

Git Push

Git Out

Creating a repo

- A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository.
- Repository is NOT a project folder. Repository is s a *data structure that stores metadata for a set of files or directory structure*
- Command to inicialize a repo in your current folder

```
git init
```

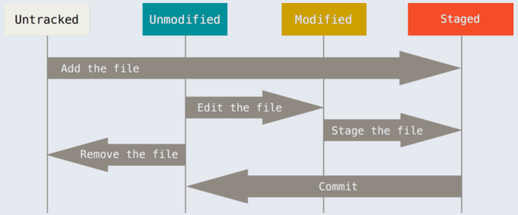
- use git add command to cpecify files you want to track, followed by git commit - add a cpecifick message to your commit

```
git add *.sh
git add .gitignore
git commit -m "add gitignore file; add scripts for some task"
```

- How to write correct commit messages is another art, but remember to write meaningdull messages
- So at this point, we have a Git repository in our project directory with tracked files and an initial commit

Changes to the repo

- At this point you have a git repo with scripts and some files
- Each file in your working directory can be in one of two states: tracked or untracked
- Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged
- In short, tracked files are files that Git knows about
- Untracked files are everything else
- Use git status to check the status of each file in current directory
- Files in a .gitignore are ignored by git repo



Manipulations with repo

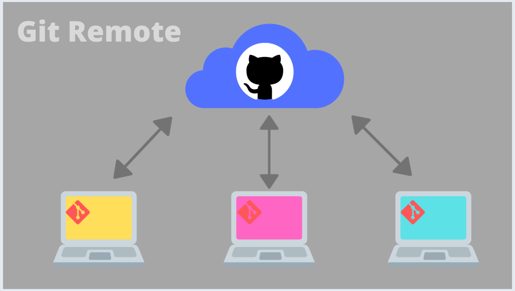
- As far as git is a decenralized system, you already have your repo with all version control features
- But now about the most powerful git feature and why we use it - remote repo
- You can either clone existing repo, or add a remote to local one
- To remove a file from both repo and directory, use git rm
- To rename a file, use git mv
- To add a remote to your repo, use git remote add <name> <url>
- To push your updates to remote, use git push <remote> <branch>

Manipulations with repo (examples)

```
mkdir test_directory
cd test_directory
echo "empty_readme" >> README.md # initialize new readme
git init
git add README.md
git commit -m "initialized new repo with readme"
# go to https://github.com, create a new repo
# go back to terminal and push your changes to remote
git remote add origin git@github.com:username/reponame.git
git commit -m "add gitignore file; add scripts for some task"
git push origin master
```

Remotes

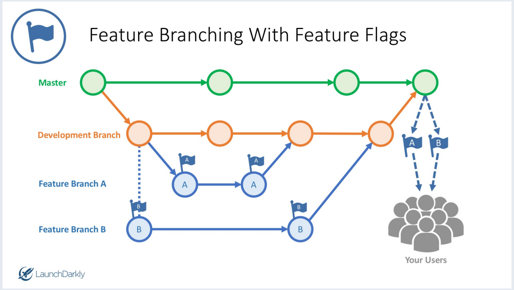
- Remote - link to remote "versions" of your repository. They are stored on some kind of service (GitHub, GitLab, BitBucket etc.) or on private server
- You can think of them as global versions of your repos (your repo can be stored on few of them simultaneously)
- To see your remotes use git remote show , by default there is only origin
- For more info on a specific remote, use git remote show remotename
- Use git pull to make a code from local up to date with origin



Branches

Branches

- Branch is a divergence from the one development line to fulfill some specific purpose
- For example - stable branch (usually master), development (dev), features, bug fixing
- Used for concurrent work of different developers on the same project
- Helps to avoid often conflicts while working on fairly independent parts
- Easier to access code for a specific feature and explore its history



Branches (examples)

```
# we already have a test_dir. continue working with it
cd test_directory
ls
> README.md
git checkout -b dev # same as git branch dev; git checkout dev
echo 'print("Hello world")' >> new.py
git add new.py
git commit -m "add new.py file"
git push origin dev
git checkout master # checkout back to master
ls
> README.md
```

You are back at master branch where there is no new.py file, it is on the dev branch
Now it is possible to merge with

```
git merge dev # about merge hell read yourself
```

GitHub Presents:
GAME OFF

GitHub

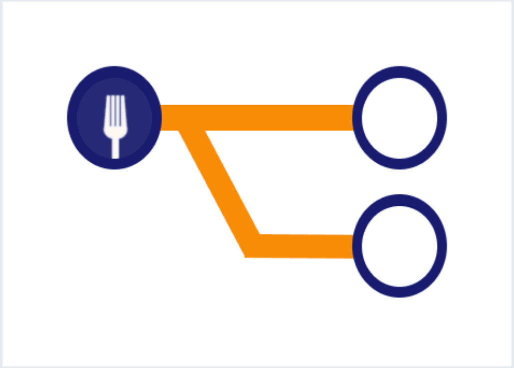
- GitHub repository - remote “versions” of your repository, it’s origin , but also much more...
- GitHub provide users with such a nice tools as Issues, Pull requests, Forks, Actions , also followers and followings, their activity, stars for projects, Projects tool for managing project development workflow, Marketplace, Insights(some statistics), Wiki, even Discussions
- So developers do a GitHub as a big
- Starting with 2021, there is such thing as GitHub CLI !

Issues

- If you use some app stored on GitHub and it works with bugs
- You have two options - report the bug or fix it and send a bugfix to maintainers
- For the first thing Issues could be helpfull (For the second - Pull requests)
- All github users can create issue on some projects pages, comment it, react on it with some emoji etc
- Issue itself - a conversation starting with user's message, that (for example) he found a bug, and some information about how you got this bug
- Also there can be feature request bugs or even improvements or other. There are dozens of such labels
- Often Issues either has a sollution writen as the last message or a pull request that solves the problem
- After that Issue can be marked as solved
- Issues are part of developement workflow, so it's imortant to keep them as much as keep source code of program itself and it's commit history

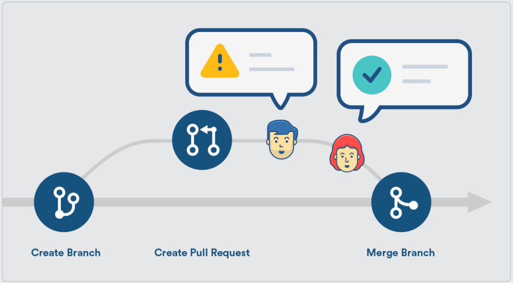
Fork

- A GitHub fork is a copy of a repo that sits in your account rather than the account from which you forked
- Once you have forked a repo, you own your forked copy
- This means that you can edit the contents of your forked repository without impacting the parent repo
- Forked repo can be detached (in theory, but it is not possible for now. By hands, support team can do that) to became an independant project



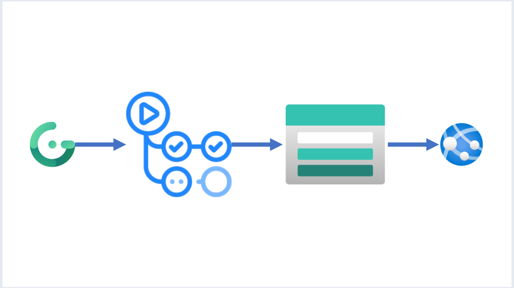
Pull request

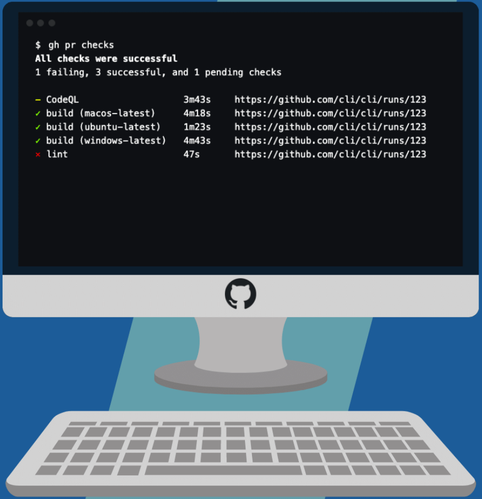
- In most cases you don’t want strangers to brake all your code, so only few people (contributors) have rights for changing the repo
- Instead, they should fork , make a new branch, implement some feature and make a pull request about merging their work to your project
- The same can be done between simple branches (but that is not necessary)
- Sometimes there is a CONTRIBUTION.md describing how to make a correct changes with clear commits and acceptable PR message for changes to be merged



GitHub Actions. CI from GitHub

- ;





```
$ gh pr checks
All checks were successful
1 failing, 3 successful, and 1 pending checks

- CodeQL                  3m43s    https://github.com/cli/cli/runs/123
✓ build (macos-latest)    4m18s    https://github.com/cli/cli/runs/123
✓ build (ubuntu-latest)   1m23s    https://github.com/cli/cli/runs/123
✓ build (windows-latest)  4m43s    https://github.com/cli/cli/runs/123
× lint                    47s      https://github.com/cli/cli/runs/123
```

THE ESSENTIAL GITHUB CLI COMMANDS

Sources

Sources

- Version control systems comparison
- Why Git is Better than X
- Git Wiki
- GitHub Wiki
- GitLab history
- GitHub documentation
- Git documentation