Git your things done!

Pavla Kratochvílová
Lukáš Růžička

February 11th, 2020

Why Git?

What if I were to work on a project that would require various versions?
With **version control** I could:

- develop multiple versions
- have a full history of changes
- access previous versions
- revert to a previous version
- synchronize among multiple computers

What is Git?

- the most widely used modern version control system
- open source
- actively maintained
- developed by Linus Torvalds
- distributed

Installation

- Fedora: sudo dnf install git
- Debian: sudo apt-get install git
- Mac: http://git-scm.com/download/mac
- Windows: http://git-scm.com/download/win

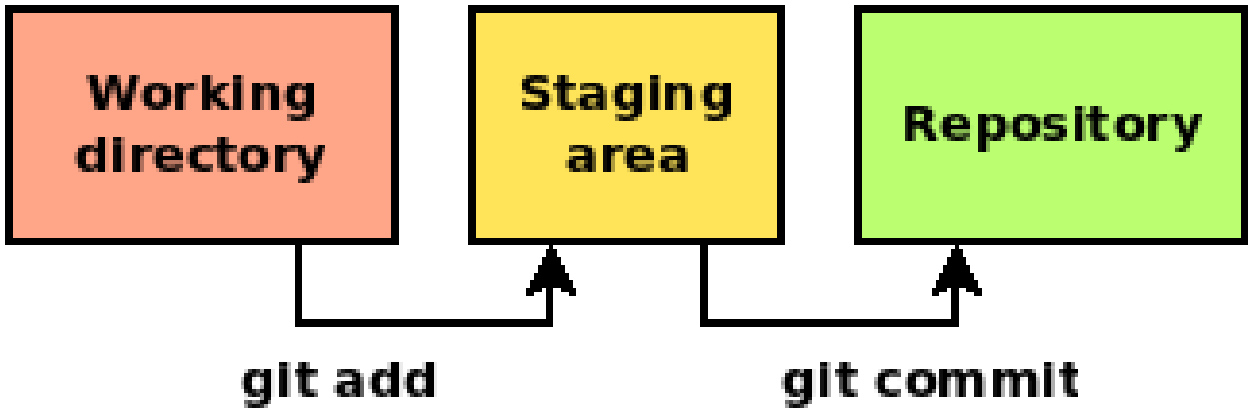
Git forge account

- Go to www.github.com.
- Create an account.

Configuration

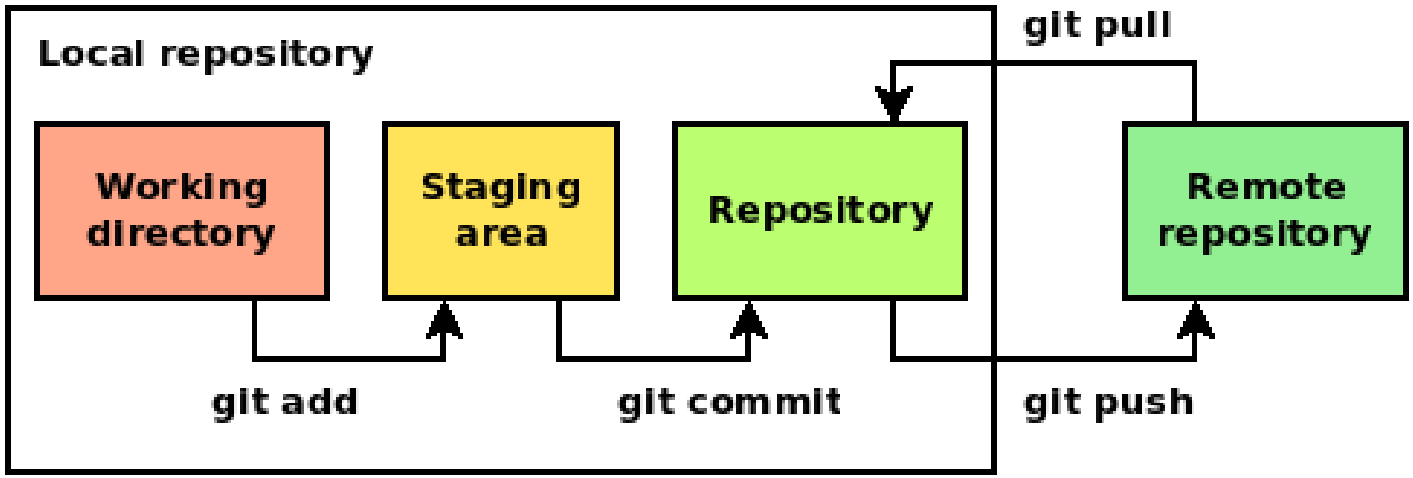
- `git config --global user.name "Your Name"`
- `git config --global user.email "yourname@example.com"`

Basics



init	create new Git repository
status	display the status quo of your local repository
add	add changes to the staging area
commit	save the changes into the tree

Remote repository



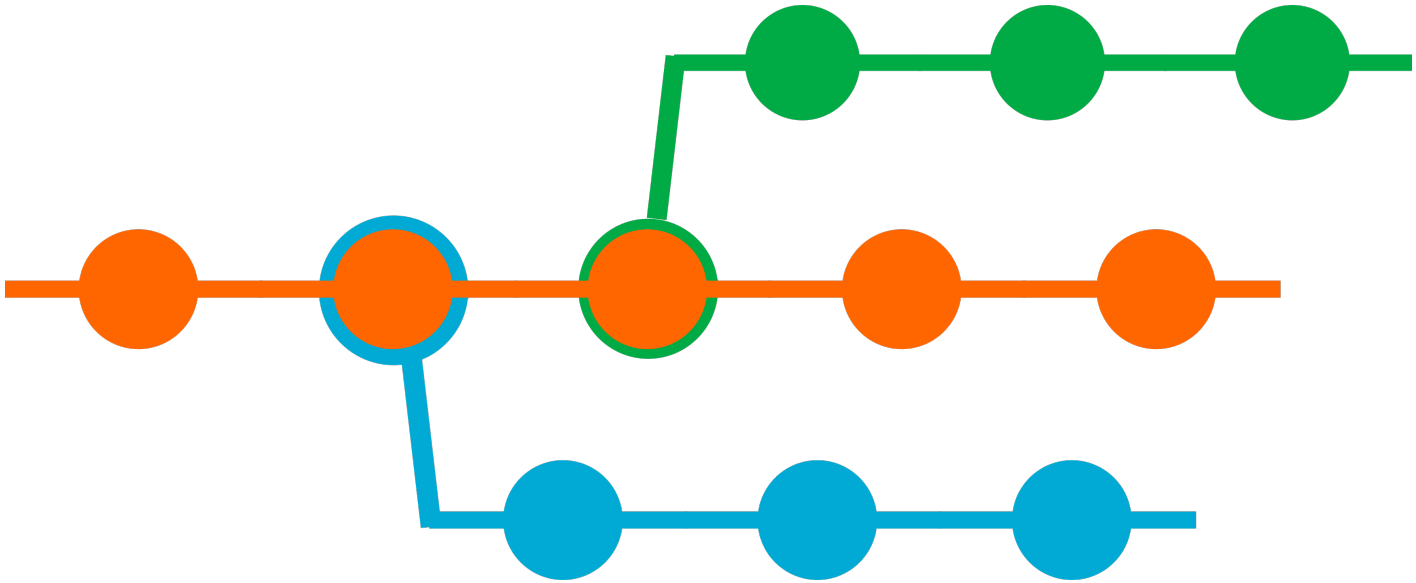
push send local changes to the server

fetch download newest changes from server

Terminology 1

clone	create a local copy of the repository
fork	create a personal copy of the repository on the server
tree	a chronological map of changes
branch	a subtree (mostly used for development)

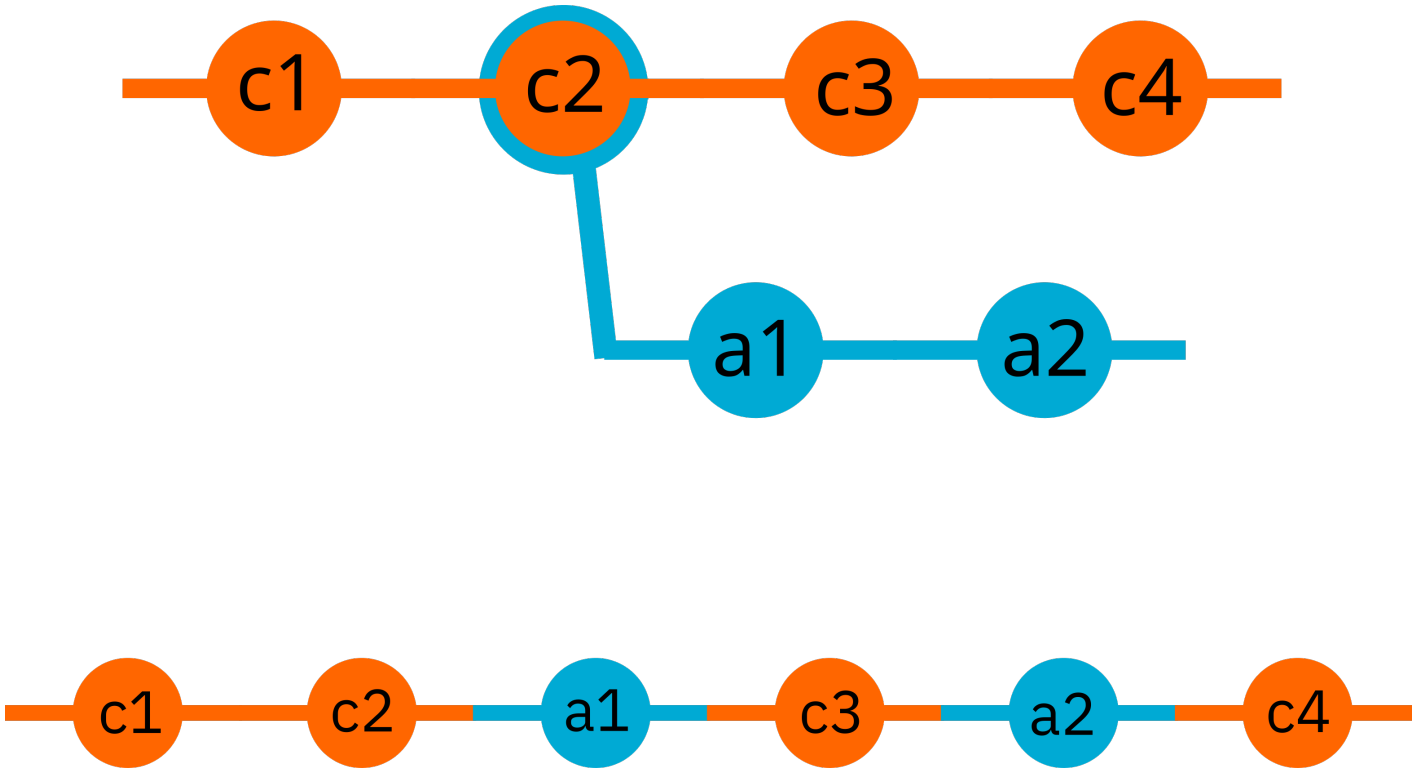
Tree and Branches



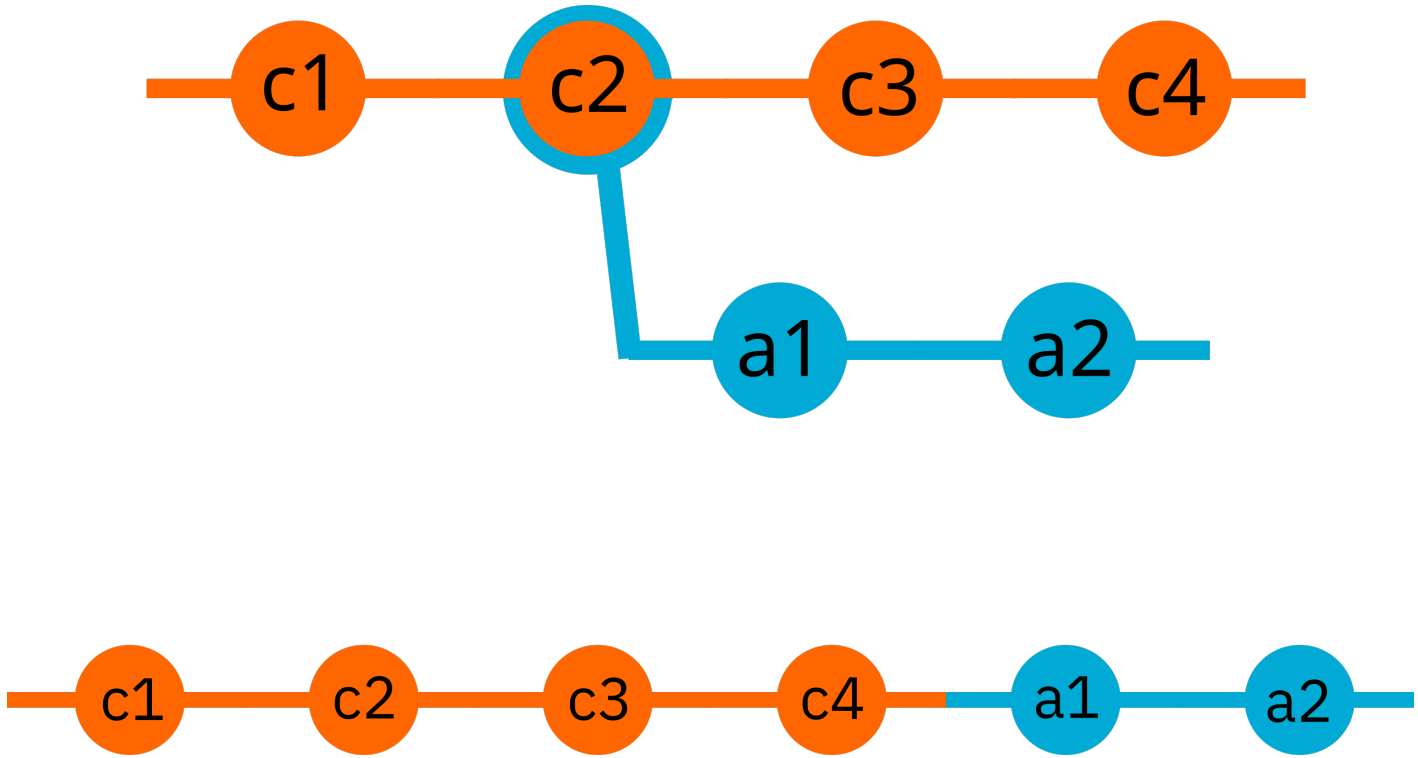
Terminology 2

merge	merge two different branches together and keep chronological order
rebase	put a branch on top of another
pull	a shortcut for *fetch* and *merge*
conflict	a problem that blocks merging of changes
squash	put two (or more) commits into one
blame	display the author of a change

How does merging work?



How does rebasing work?



Fork the repository

- creates a server-based copy of the repo
- go to your Git forge webUI
- push the **Fork** button

Clone the repository

- creates a local copy of the repository in a new directory
- `git clone <repo-address>`
- `git clone <repo-address> <directory>`

Task 1

1. As a group, fork repository `https://github.com/dokumentarista/trygit.git`.
2. Set up commit rights for your members.
3. Clone the fork to your machine.
4. Go to that directory.
5. Display its content (`ls -a`)

Developing the project (adding changes)

1. open, edit, save files as you would normally do
2. see the new status
   - `git status`
3. add files you want git to start tracking
   - `git add`
4. save the changed files into the git tree
   - `git commit -m "Explain why"`
5. synchronize your git tree with the server version
   - `git push`

Task 2

Although a group, work individually

1. Open the `names.txt` file in the repo
2. Add your name to the list of names
3. Commit your changes
4. Push them onto the server

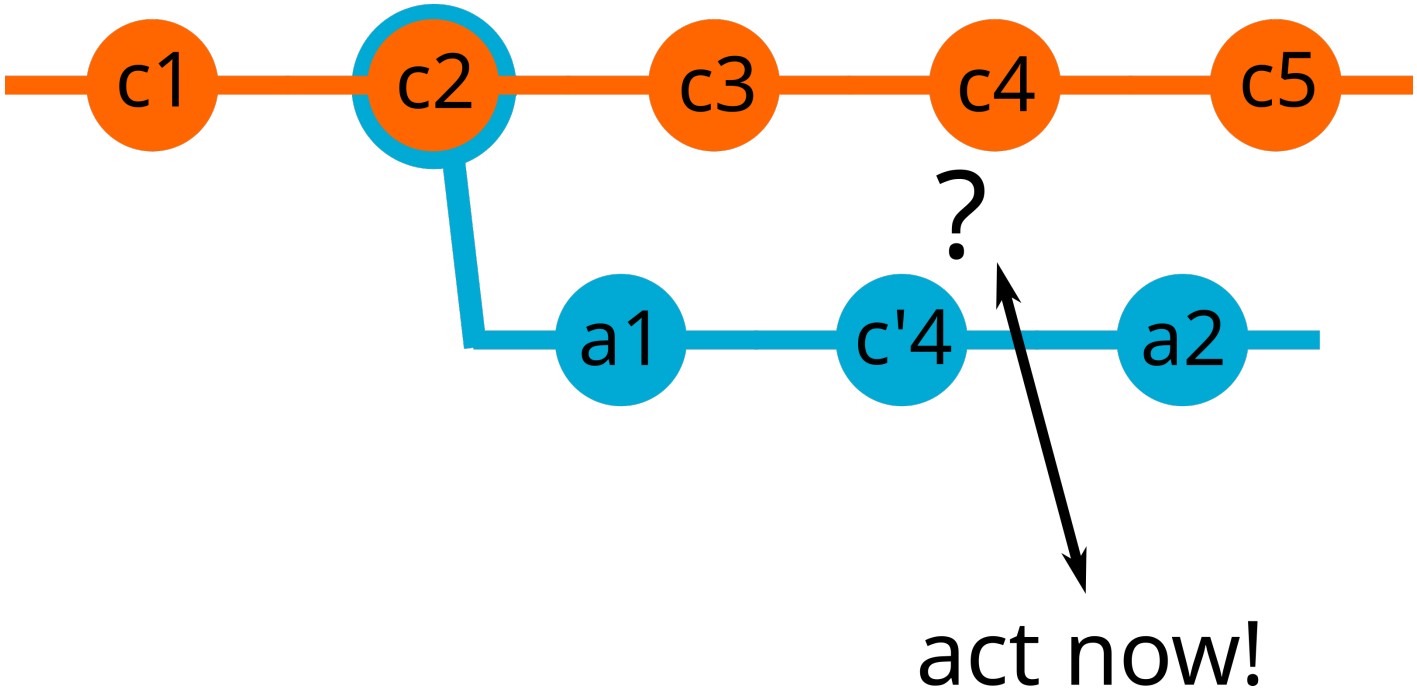
Getting the first conflict

Git conflict, sometimes referred to as **merge conflict**, happens when:

- two (or more) versions of one change
- at the same time

When in conflict, you cannot work with the remote repository because Git protects your data from being damaged.

When do I get a merge conflict?



When you try to push

```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://github.com/dokumentarista/trygit.git'
hint: Updates were rejected because the remote
hint: contains work that you do
not have locally.
hint: This is usually caused by another repository
hint: pushing
to the same ref. You may want to first
hint: integrate the remote changes
(e.g., 'git pull ...')
hint: before pushing again.
hint: See the 'Note about fast-forwards' for details.
```

When you try to pull

```
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From https://github.com/dokumentarista/trygit
34c12d6..d8a0bea  master     -> origin/master
Auto-merging names.md
CONFLICT (content): Merge conflict in names.md
Automatic merge failed; fix conflicts and then
commit the result.
```

In the file

```
# Names of login names.

<<<<<<< HEAD
### Add your login name to the last available slot.

1. pkratoch
=======
### Add your login name to the last available slot.

1. lruzicka
>>>>>>> d8a0beae9626d523d509b9fc53de06c435999d24
2.
3.
4.
5.
```

How to solve merge conflicts?

- Open the conflicting file.
- Explore the marked area.
- Your changes are marked **HEAD** above the division line.
- `========` is the division line.
- Remote changes are bellow the division line.
- Rewrite the file as you want it to be and save it.
- `git add corrected-file`.
- `git merge --continue`
- Edit the commit message if asked.

Conflict fixed

```
# Names of login names.

### Add your login name to the last available slot.

1. pkratoch
2. lruzicka
3.
4.
5.
```

How to limit conflicts?

- Work in **branches**.
- **Fork** the project and work in your version.
- Plan ahead.
- Communicate.

Conflicts will always happen, love them, nurture them and fix them carefully.

What is a branch?

- alternate development version
- it checks out from a certain commit
- it can branch from **master** or another branch
- it typically diverges from its origin very quickly
- it allows you to work individually without having to solve many conflicts as you go

How to work in a branch?

- Create a new branch (`git checkout -b new`)
- Write your changes there.
- Fix merge conflicts if any.
- Merge or rebase possible changes in the original branch to your branch to make it merge ready.
- Have it merged (or rebased) back into its origin.

git merge

- Merges two branches into one.
- The checked-out branch will be altered.
- It keeps track of history.
- It is chronological.
- It produces a **merge message**

git rebase

- Merges two branches into one.
- The checked-out branch will be altered.
- It does not keep track of history.
- It is not chronological.
- It accepts foreign commits, merges them to your branch, and puts your commits on top of that.
- It helps to keep the history of the master branch free from merge commits.

Task 3

1. Delete the repo files and clone it again.
2. Each person in the group creates their own branch.
3. Communicate with the team.
4. Add your name to the list of names in your branch.
5. Merge or rebase the original branch onto your branch.
6. Fix conflicts.
7. Have it merged.

Task 4

1. As a group, fork repository `https://github.com/dokumentarista/crossword.git`.
2. Set up commit rights for your members.
3. Clone the repo.
4. Solve the crossword.

What if I don't have access to repository?

- Very common in open source world
- Send patch via email?

Fork workflow

- Fork a repository
- Clone a repository
  - `git clone <repo-address> <directory>`
- See remote repositories
  - `git remote -v`
- Add the other remote repository
  - `git remote add <name> <repo-address>`
- Make changes and push them to your fork
  - `git push -set-upstream <remote> <branch>`
- Make pull request

Changing the history – interactive rebase

- Can change commit messages.
- Can merge two (or more) commits – squash them.
- Can throw away commits.
- Makes severe changes to the repo structure – risky.
- It changes the fundaments for your collaborators.
- Needs to be force pushed.
- Should only be done in individual branches.

How to recover from interactive rebase?

- Checkout the branch.
- Fetch the new repo data
  - `git fetch origin`
- Rebase your branch onto the original branch.
  - `git rebase origin/master`
- All changes from your branch will appear on top of the original branch.
- Alternatively, you can use an option that will do the rebase for you, if possible.
  - `git pull --rebase`
- **Merging the branch would never work**, because the history has been changed.

Undo local changes – reset

- You can reset the HEAD to a previous commit.
- You can either use hashes or `HEAD~3`
- You can use **soft**, **mixed** or **hard** reset.
- Default is mixed – it changes the HEAD marker and unstages files, but leaves them untouched.
- Hard reset will delete your files – think twice.
- The operation goes back in history – needs rebasing.
- All changes can be recovered until you push to the server.
- Should only be done in individual branches.

Undo local changes – revert

- You can revert to a previous commit.
- You can either use hashes or `HEAD~3`
- A new commit will be added, that undoes the changes.
- The operation does not go back in history, can be forwarded.
- All changes can be recovered any time locally.
- Can be done in cooperative branches.

Questions?

If you have any questions, just ask now . . .

. . . or hold it forever.

Thank you for your attention and have a great day!