Chair of Connected Mobility
TUM School of Computation, Information and Technology
Technical University of Munich

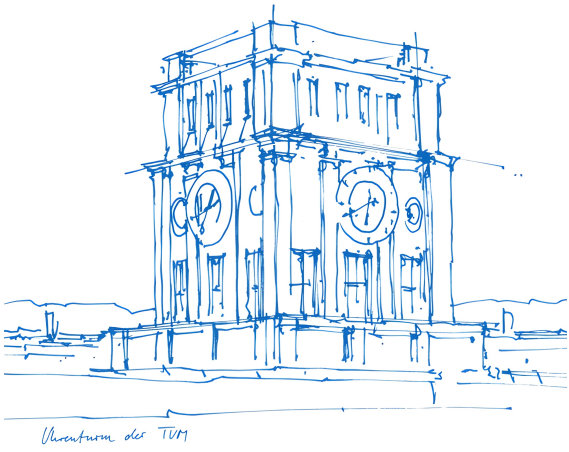
Open Source Lab

Git Basics and Getting Started

Fabian Sauter, Christian Menges

Chair of Connected Mobility
TUM School of Computation, Information and Technology
Technical University of Munich

Garching, October 16, 2024



Outline

Credits

These slides are based on the awesome materials from:

Moritz Sichert (sichert@in.tum.de)
Michael Freitag (freitagm@in.tum.de)
} Systems Programming in C++ (Practical Course)

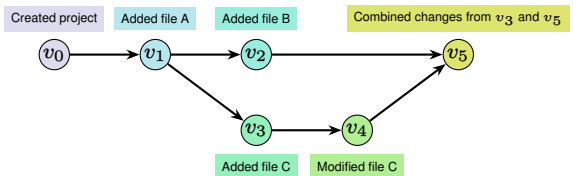
Pro Git book: https://git-scm.com/book

Version Control Systems (VCS)

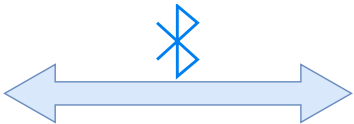
- Code projects evolve gradually
- Incremental changes, also called *versions*, should be tracked to allow:
  - Documentation of the project history
  - Selective inspection/modification of specific versions
  - Efficient collaboration when working in a team
- A **V**ersion **C**ontrol **S**ystem (VCS) manages versions, usually represent them in a directed acyclic graph

Version Control Systems (VCS)

- Code projects evolve gradually
- Incremental changes, also called *versions*, should be tracked to allow:
  - Documentation of the project history
  - Selective inspection/modification of specific versions
  - Efficient collaboration when working in a team
- A **V**ersion **C**ontrol **S**ystem (VCS) manages versions, usually represent them in a directed acyclic graph



A more concrete example. . .



A more concrete example...

```
int mo6312r(int i, int i2, int i3, int i4) {
  byte[] bArr = arr1;
  int i5 = i2 >> 4;
  return mo6308n(mo6308n((bArr[mo6308n((((arr2[mo6308n((((bArr[mo6308n((i + i2) + i3) % 16] + i4) +
       mo6308n(i5)) - i2) - i3) % 16] + i3) + i2) - i4) - mo6308n(i5)) % 16] - i2) - i3) % 16);
}
```

A more concrete example...

```
int mo6312r(int i, int i2, int i3, int i4) {
  byte[] bArr = arr1;
  int i5 = i2 >> 4;
  return mo6308n(mo6308n((bArr[mo6308n((((arr2[mo6308n((((bArr[mo6308n((i + i2) + i3) % 16] + i4) +
       mo6308n(i5)) - i2) - i3) % 16] + i3) + i2) - i4) - mo6308n(i5)) % 16] - i2) - i3) % 16);
}
```

```
int mo6308n(int i) {
while (i > 255) {
  i += InDevComp.SOURCE_ANY; // -256
}
while (i < 0) {
  i += 256;
}
  return i;
}
```

- Looks like a fancy `mod256` for Java.
- So let's replace it...

A more concrete example...

```
int mo6312r(int i, int i2, int i3, int i4) {
  byte[] bArr = arr1;
  int i5 = i2 >> 4;
  return mod256(mod256((bArr[mod256((((arr2[mod256((((bArr[mod256((i + i2) + i3) % 16] + i4) + mod256(i5
      )) - i2) - i3) % 16] + i3) + i2) - i4) - mod256(i5)) % 16] - i2) - i3) % 16);
}
```

A more concrete example...

```
int mo6312r(int i, int i2, int i3, int i4) {
  byte[] bArr = arr1;
  int i5 = i2 >> 4;
  return mod256(mod256(bArr[mod256((((arr2[mod256(((bArr[mod256(i + i2 + i3) % 16] + i4) + mod256(i5) -
       i2) - i3) % 16] + i3) + i2) - i4) - mod256(i5)) % 16] - i2 - i3) % 16);
}
```

- Let's continue simplifying and remove parentheses...

A more concrete example...

```
uint8_t shuffle(int dataNibble, int nibbleCount, int keyLeftNibbel, int keyRightNibbel) {
  uint8_t i5 = mod256(nibbleCount >> 4);
  uint8_t tmp1 = numbers1[mod256(dataNibble + nibbleCount + keyLeftNibbel) % 16];
  uint8_t tmp2 = numbers2[mod256(tmp1 - keyRightNibbel + i5 - nibbleCount - keyLeftNibbel) % 16];
  uint8_t tmp3 = numbers1[mod256(tmp2 + keyLeftNibbel + nibbleCount - keyRightNibbel - i5) % 16];
  return mod256(tmp3 - nibbleCount - keyLeftNibbel) % 16;
}
```

- And now split it up and rename everything...

A more concrete example...

```
uint8_t shuffle(int dataNibble, int nibbleCount, int keyLeftNibbel, int keyRightNibbel) {
  uint8_t i5 = mod256(nibbleCount >> 4);
  uint8_t tmp1 = numbers1[mod256(dataNibble + nibbleCount + keyLeftNibbel) % 16];
  uint8_t tmp2 = numbers2[mod256(tmp1 - keyRightNibbel + i5 - nibbleCount - keyLeftNibbel) % 16];
  uint8_t tmp3 = numbers1[mod256(tmp2 + keyLeftNibbel + nibbleCount - keyRightNibbel - i5) % 16];
  return mod256(tmp3 - nibbleCount - keyLeftNibbel) % 16;
}
```

- And now split it up and rename everything...

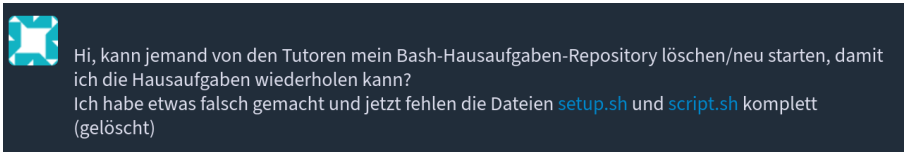
And then we notice, that our tests do not produce the same results as the Java code anymore.

What do we do now?
Reset and start over?
Spend hours debugging?

Another example

Hi, kann jemand von den Tutoren mein Bash-Hausaufgaben-Repository löschen/neu starten, damit ich die Hausaufgaben wiederholen kann?
Ich habe etwas falsch gemacht und jetzt fehlen die Dateien setup.sh und script.sh komplett (gelöscht)

Another example



He deleted his code and asks us whether we could reset it for him, so he could start over again.
But since he uses git, he can go back to any version of his code by himself.

- Many VCS exist, Git is a very popular one: Used by projects like Linux, GCC, LLVM, etc.
- Git, in particular, has the following advantages compared to other version control systems (VCS):
  - Open source (LGPLv2.1)
  - Decentralized, i.e. no server required
  - Efficient management of *branches* and *tags*
- All Git commands are documented with man-pages (e.g. type `man git-commit` to see documentation for the command `git commit`)
- Pro Git book: https://git-scm.com/book
- Git Reference Manual: https://git-scm.com/docs

Git History

- Initiator Linus Torvalds
- Goals speed, ”simple” design, fully distributed, able to handle large projects, . . .
- git is British slang for ”pig headed, think they are always correct, argumentative”
- Quoting Linus ”I’m an egotistical bastard, and I name all my projects after myself. First ’Linux’, now ’Git’.”[1]

[1] https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27Git.27_name.3F
[2] https://marc.info/?l=git&m=117254154130732

Git History

- Initiator Linus Torvalds
- Goals speed, ”simple” design, fully distributed, able to handle large projects, . . .
- git is British slang for ”pig headed, think they are always correct, argumentative”
- Quoting Linus ”I’m an egotistical bastard, and I name all my projects after myself. First ’Linux’, now ’Git’.”[1]

- 1991-2002 Changes were passed as patches and archived files.
- 2002 The Linux kernel project began using a proprietary DVCS called BitKeeper.
- April 2005 The relationship between the commercial company behind BitKeeper and the Linux Kernel community broke down.
  - SourcePuller was created by reverse engineering the BitKeeper protocols.
  - Free use license got withdrawn.

⇒ Linus Torvalds started working on an alternative, called git.

- 7. April 2005 git is self-hosted[2]

[1] https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27Git.27_name.3F
[2] https://marc.info/?l=git&m=117254154130732

Git Concepts

Tree: A collection of files (not directories!) with their path and other metadata. This means that Git does *not* track empty directories.

Git Concepts

Tree: A collection of files (not directories!) with their path and other metadata. This means that Git does *not* track empty directories.

Commit: A snapshot of a *tree*. Identified by a SHA1 hash. Each commit can have multiple parent commits. The commits form a directed acyclic graph.

Git Concepts

Tree: A collection of files (not directories!) with their path and other metadata. This means that Git does *not* track empty directories.

Commit: A snapshot of a *tree*. Identified by a SHA1 hash. Each commit can have multiple parent commits. The commits form a directed acyclic graph.

Branch: A named reference to a *commit*. Every repository usually has at least the `master` (`main`) branch and contains several more branches, like `fix-xyz` or `feature-abc`.

Git Concepts

Tree: A collection of files (not directories!) with their path and other metadata. This means that Git does *not* track empty directories.

Commit: A snapshot of a *tree*. Identified by a SHA1 hash. Each commit can have multiple parent commits. The commits form a directed acyclic graph.

Branch: A named reference to a *commit*. Every repository usually has at least the `master` (`main`) branch and contains several more branches, like `fix-xyz` or `feature-abc`.

Tag: A named reference to a *commit*. In contrast to a branch a tag is usually set once and not changed. A branch regularly gets new commits.

Git Concepts

Tree: A collection of files (not directories!) with their path and other metadata. This means that Git does *not* track empty directories.

Commit: A snapshot of a *tree*. Identified by a SHA1 hash. Each commit can have multiple parent commits. The commits form a directed acyclic graph.

Branch: A named reference to a *commit*. Every repository usually has at least the `master` (`main`) branch and contains several more branches, like `fix-xyz` or `feature-abc`.

Tag: A named reference to a *commit*. In contrast to a branch a tag is usually set once and not changed. A branch regularly gets new commits.

Repository: A collection of Git objects (*commits* and *trees*) and references (*branches* and *tags*).

Creating a Git Repository

Create a new directory (home) for our repository and change into it.

```
mkdir myRepo && cd myRepo
```

Initialize a new Git repository.

```
git init
```

Creating a Git Repository

Create a new directory (home) for our repository and change into it.

```
mkdir myRepo && cd myRepo
```

Initialize a new Git repository.

```
git init
```

Set the name that will be used when creating a commit.*

```
git config --global user.name "Firstname Lastname"
```

Set the e-mail address that will be used when creating a commit.*

```
git config --global user.email "first.last@example.org"
```

Shows the current status and information for this repository.

```
git status
```

*Required only for the first time you create a Repository.

Creating a Git Repository

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "first.last@example.org"
$ mkdir myRepo && cd myRepo
$ git init
Initialized empty Git repository in /tmp/myRepo/.git/
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
$ ls -la # Show the contents of the directory
total 0
drwxrwxr-x. 3 user  user   60 Sep 7 13:16 .
drwxrwxrwt. 31 root root  780 Sep 7 13:35 ..
drwxrwxr-x. 7 user  user  200 Sep 7 13:16 .git
```

Cloning an Existing Git Repository

Usually, we do not want to start a new repository, instead, we want to contribute to an existing one.
Creating a local copy (cloning) some remote repository.

```
git clone <remote>
```

Example cloning the `cpr` repository from GitHub.

```
git clone https://github.com/libcpr/cpr.git
Cloning into 'cpr'...
remote: Enumerating objects: 4969, done.
remote: Counting objects: 100% (557/557), done.
remote: Compressing objects: 100% (309/309), done.
remote: Total 4969 (delta 324), reused 356 (delta 224), pack-reused 4412
Receiving objects: 100% (4969/4969), 1004.03 KiB | 3.16 MiB/s, done.
Resolving deltas: 100% (3300/3300), done.
```

Branches

A branch is a named reference to a specific commit.

Gives you a list of all (local) branches which is currently active.

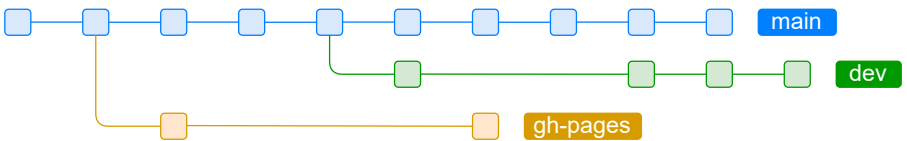
```
git branch [-r]
```

Create a new branch from the current commit.

```
git branch <name>
```

Switch to another existing local branch, i.e. change all files in the working directory so that they are equal to the tree of the other branch.

```
git switch <name>
```



Tags

A named reference to a commit. In contrast to a branch, a tag is usually set once and not changed. A branch regularly gets new commits.

Gives you a list of all (local) tags.

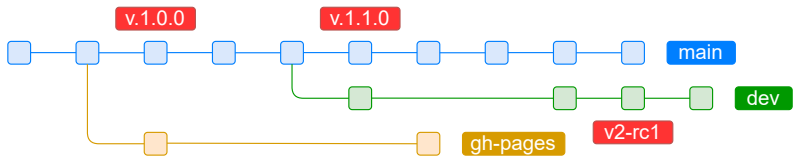
```
git tag
```

Create a new tag from the current commit. With the "-s" option you can sign it using a PGP key.

```
git tag [-s] <name>
```

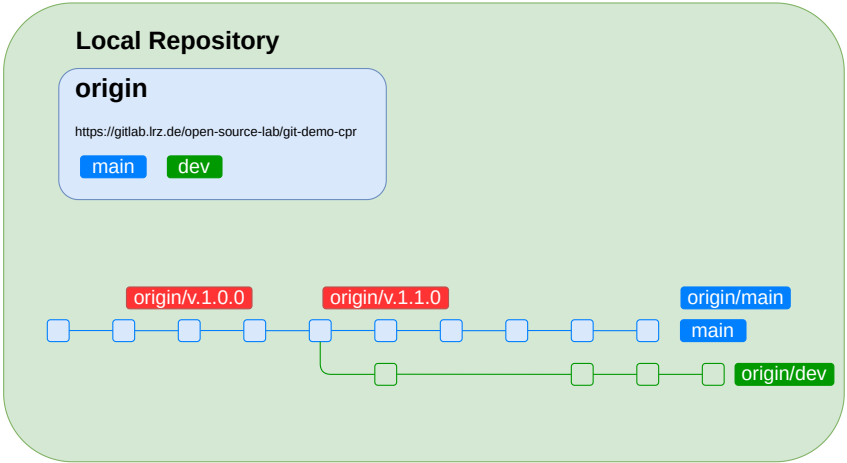
Checkout the given tag. Be aware to make changes you need to create a branch first!

```
git checkout <tag>
```



Adding a Remote to a Local Repository

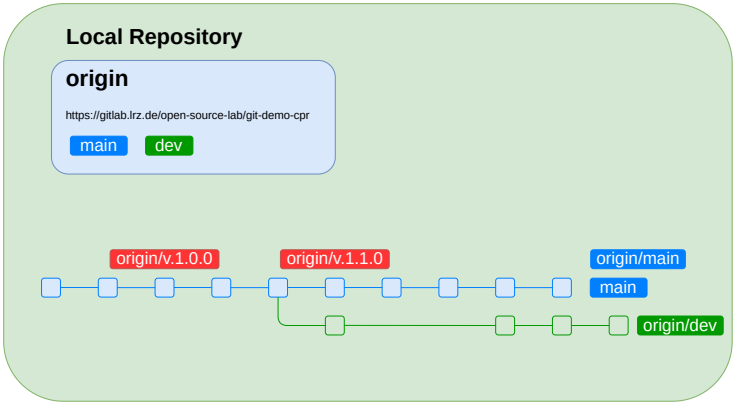
After cloning a repository from a remote, there is one remote called `origin` by default. There is usually a single local branch (`main`).



Adding a Remote to a Local Repository

It is also possible to add new remotes. Useful for rebasing after forking.

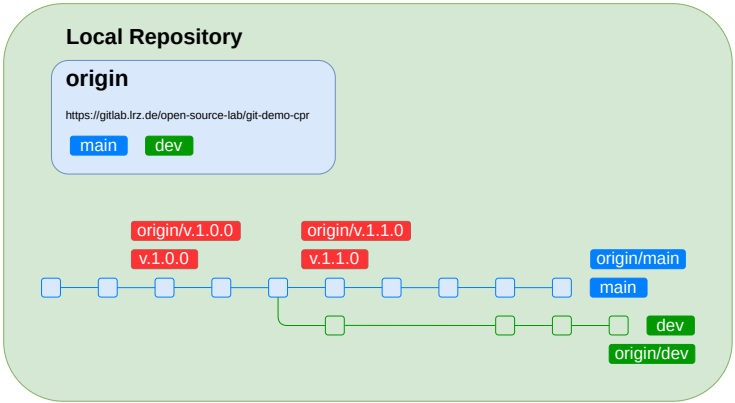
```
git remote add <myNewRemoteName> <url>
```



Adding a Remote to a Local Repository

It is also possible to add new remotes. Useful for rebasing after forking.

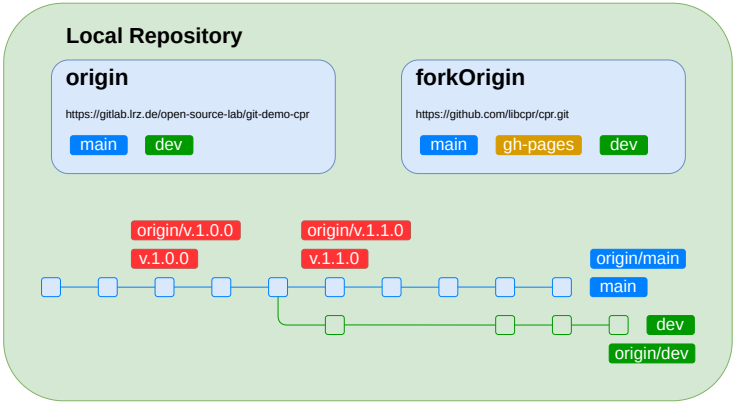
```
git remote add <myNewRemoteName> <url>
```



Adding a Remote to a Local Repository

Example:

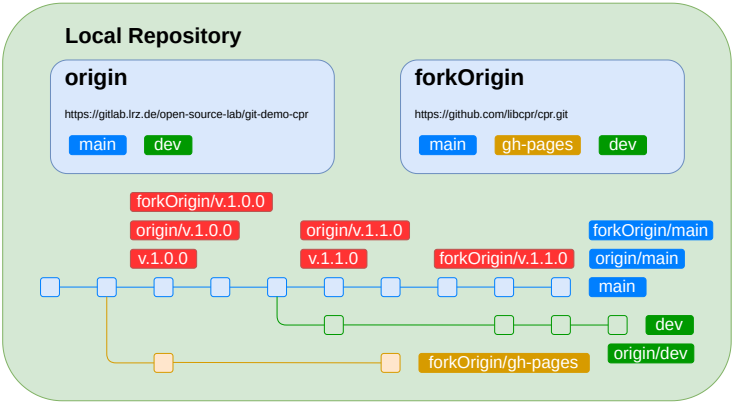
```
git remote add forkOrigin https://github.com/libcpr/cpr.git
```



Adding a Remote to a Local Repository

Example:

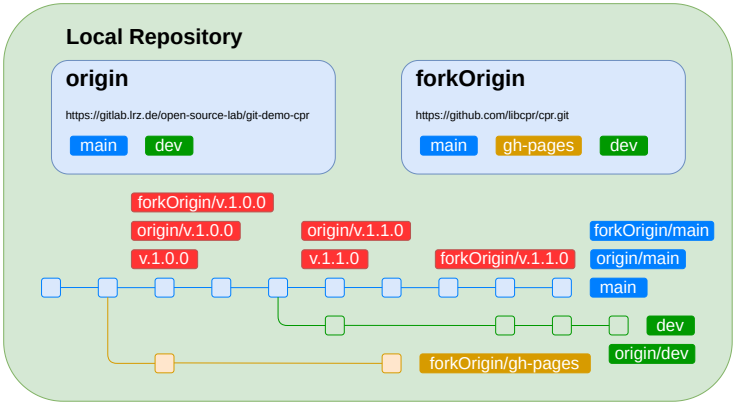
```
git remote add forkOrigin https://github.com/libcpr/cpr.git
```



Adding a Remote to a Local Repository

Don’t forget to fetch branches and tags from the remote after adding them.

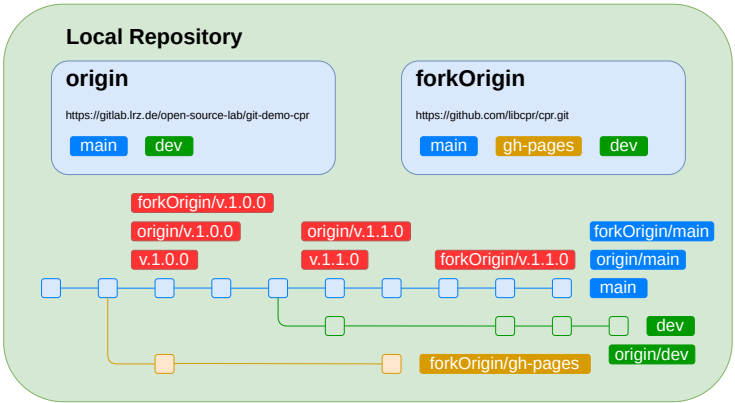
```
git fetch <remote>
```



Adding a Remote to a Local Repository

Example:

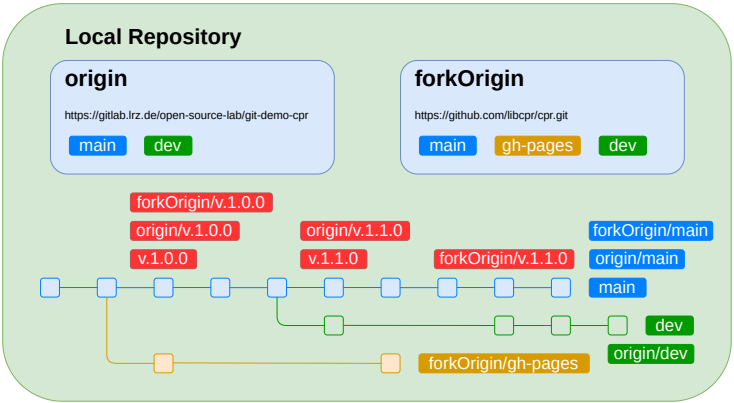
```
git fetch forkOrigin
```



Adding a Remote to a Local Repository

It is also possible to specify the remote to push/pull from.

```
git push <remote> <branch>
```



Exercise

- Clone: https://gitlab.lrz.de/open-source-lab/git-demo-cpr
- Checkout the branch: "1.10.x"
- Create a new branch called "1.10.x_yourName" based on "1.10.x".
- Switch to the new branch.

Commits

A snapshot of a *tree*. Identified by a SHA1 hash. Each commit can have multiple parent commits. The commits form a directed acyclic graph.

Stages all changes inside the given file and starts tracking it in case it is not already being tracked.

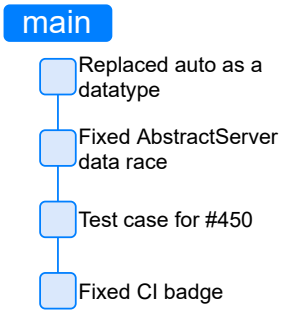
```
git add <file>
```

With this, you can bundle all your `staged` changes (`git add`) to one commit with a commit message.
The "-S" option allows you to sign commits using a PGP key.

```
git commit [-S] -m "Some message"
```

This enables signing commits by default for all your repositories.

```
git config --global commit.gpgsign true
```



Commit Messages

A few guidelines for creating commits:

Commit Messages

A few guidelines for creating commits:

Dos

- Commit early and often.
- Split up your work into atomic commits.
- Make commit messages meaningful.
- Subject should be less than 50 characters.
- Do not end the subject line with a ”.”
- Separate the subject and body by a blank line.

Commit Messages

A few guidelines for creating commits:

Dos

- Commit early and often.
- Split up your work into atomic commits.
- Make commit messages meaningful.
- Subject should be less than 50 characters.
- Do not end the subject line with a ”.”
- Separate the subject and body by a blank line.

Don’ts

Figure 1 ”Git Commit” by xkcd

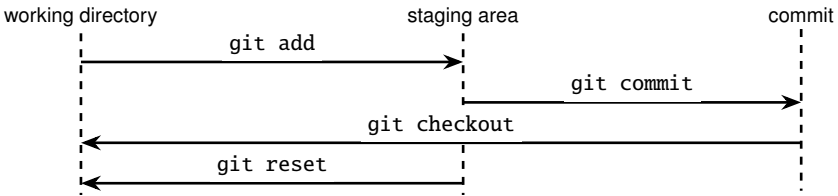
Funny, but not recommended: http://whatthecommit.com

Git Working Directory and Staging Area

When working with a Git repository, changes can live in any of the following places:

- In the working directory (when you edit a file)
- In the staging area (when you use `git add`)
- In a commit (after a `git commit`)

Once a change is in a commit and it is referenced by at least one branch or tag you can always restore (go back to) it, even if you remove the file.



History

Allows you to inspect the git commit history.
"`--online`" condenses every commit into one line.
"`--graph`" shows the git history as an ASCII graph

```
git log [--oneline] [--graph]
```

Example

```
$ git log --oneline --graph
*   bc2a09a (HEAD -> main) Merge branch 'feature-print'
|\
| * ad80c4c (feature-print) Fixed print helper new line
| * 7fce54f Added print helper framework
* | 17fdfab Added reader framework
|/
* 3783ea8 Added main call
```

Exercise

- Open the `README.md` file and replace "Fabian Sauter" with your name as a contributor.
- Create a commit with those changes.
- Take a look at the commit history using: `git log`
- **Don't push your changes yet!**

Pushing and Pulling

When working in a team, it is required to synchronize changes between the individual team members.
⇒ For this a remote repository is being used where everybody pushes its changes (commits) to.
Usually called "`origin`".

Upload the current branch to a remote repository.
⚠ "`-f`" force overrides the remote branch. Required in case you changed the git history (deleting commits/rebasing/...). Be extremely careful with this option!
"`--force-with-lease`" is a better alternative since it has the same effect, but does fail in case there are new remote changes.

```
git push [-f] [--force-with-lease]
```

Retrieve the latest metadata from `origin` and check if there are changes available.

```
git fetch
```

Fetches changes from `origin` and merges/rebases them with your local changes.

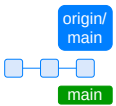
```
git pull [--rebase]
```

Enables rebasing instead of merging by default.

```
git config --global pull.rebase true
```

Push and Pull in Action

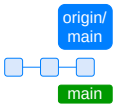
Local



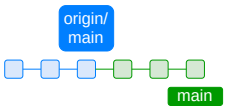
Remote

Push and Pull in Action

Local



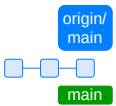
You create three new commits.



Remote

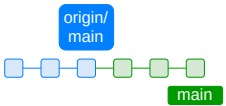
Push and Pull in Action

Local

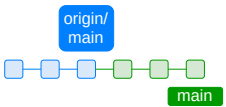


Remote

You create three new commits.

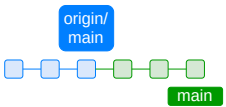


In the meantime, somebody else pushed two new commits to remote.



Push and Pull in Action

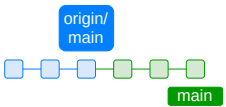
Local



Remote

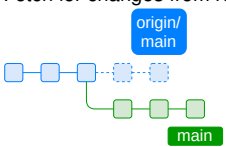
Push and Pull in Action

Local



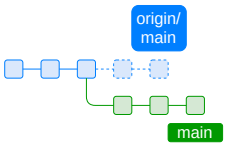
Remote

Fetch for changes from remote with `git fetch`



Push and Pull in Action

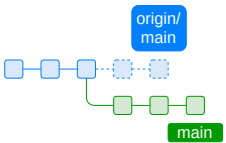
Local



Remote

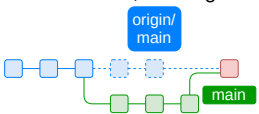
Push and Pull in Action

Local



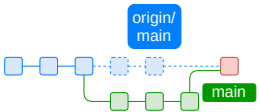
Remote

To apply changes we have found with `git fetch`, we use `git pull`
By default this will merge changes and create a so-called ”merge commit”.
In some cases, creating a merge commit is considered bad practice. More on that later...



Push and Pull in Action

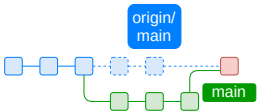
Local



Remote

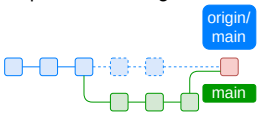
Push and Pull in Action

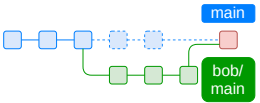
Local



Remote

To push our changes to remote, we use `git push`

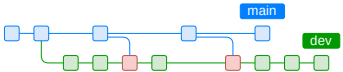




Now both, our local and remote branch have the same history and we are done.

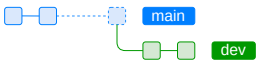
Merge and Rebase

To stay up to date with the changes in `main`, we merged them into our `dev` branch twice until now.
As a result, we got two (red) merge commits, each with its own commit message.



This is considered bad practice. Instead, we should rebase.


```
git rebase main
```



While a `merge` keeps your history, `rebase` will rewrite it for all commits in our `dev` branch!

Reverting Changes

Create a new commit that is the ”inverse” of the specified commit.

```
git revert <commit-hash>
```

Reset the current branch to the last commit. No files are changed.
”<commit-hash>” allows you to specify a specific commit to revert to.
⚠ ”--hard” resets all files in the current working directory back to the specified commit.

```
git reset [--hard] [<commit-hash>]
```

Shows a history of SHA1 commit hashes that were added or removed.
Allows restoring removed commits if they were not garbage collected yet.

```
git reflog
```

.gitignore

Allows you to specify intentionally untracked (ignored) files.
Patters:

- "!" negates the following pattern.
- "*" matches anything except a "\".
- "**" matches anything before or after the given path.

Example

```
# Build directory
build/
!build/bin/**
# Debug files
*.dSYM/
*.su
```

A collection of `.gitignore` templates: https://github.com/github/gitignore

.gitattribute

Allows you to specify attributes for paths. For example to normalize line endings.
Line format:

```
pattern attr1 attr2 ...
```

Here the `pattern` is the same as for `.gitignore` without negative patterns.

Example

```
*               text=auto
*.txt           text
*.vcproj        text eol=crlf
*.sh            text eol=lf
```

`.gitattribute` generator: https://gitattributes.io

The right Tool for the right Job

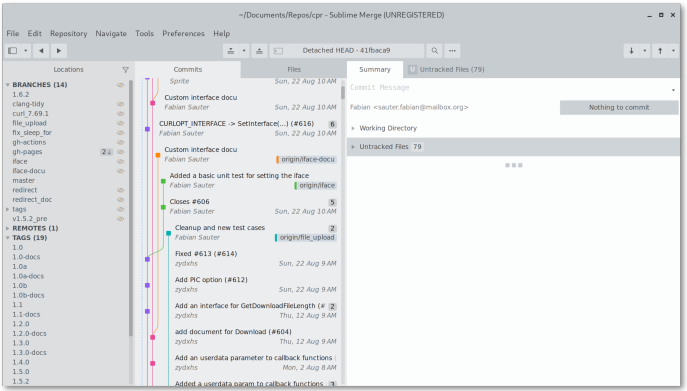
Sometimes using git in a terminal is too cumbersome or degrades to just copy and pasting complex commands from Stack Overflow...

- There exists a bunch of graphical user interfaces for git[3].
- They support you in performing ”complex” actions like merging, rebasing, and solving merge conflicts.
- Most code editors come with basic git support (like in visual studio (code) or the Jetbrains IDEs).
- Two popular alternatives:
  - Sublime Merge
  - GitKraken

[3] https://git-scm.com/downloads/guis

Sublime Merge[4]

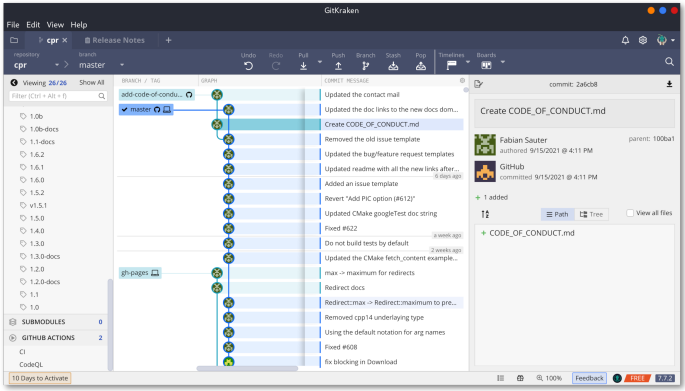
Platforms: Linux, Mac, Windows
Price: $99/user, $75 annual business sub, free eval
License: Proprietary



[4] https://www.sublimemerge.com/

GitKraken[5]

Platforms: Linux, Mac, Windows
Price: Free / $29 / $49
License: Proprietary



[5]https://www.gitkraken.com/

Summary

1. a) Create a repository

```
git init
```

1. b) Clone repository

```
git clone <remote>
```

2. Make changes
3. Add changes

```
git add <file>
```

3. Commit changes

```
git commit -m "Some message."
```

4. Done? No - Go back to 2. Yes - Continue

5. Rebase

```
git pull --rebase
```

6. Push

```
git push
```

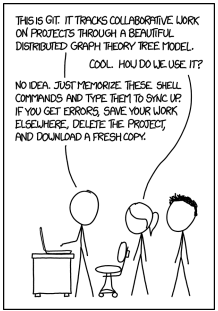


Figure 2 "Git" by xkcd

Outline

Credits

These slides are based on the awesome materials from:

The Open Source Development Course

https://github.com/crocs-muni/open-source-development-course

There, they go far deeper than we have time for in a practical course. We focus more on the practical parts instead of the theoretical ones.
BUT in case you want a deeper look into everything we are covering check out this awesome course.

Getting Started

Now that you've started yourself, we can start again together...

1. Selecting a project/issue
2. Checking if the project is still alive
3. Checking the contribution guidelines
4. Forking and starting to work

Selecting a Projects and Issue

The best project is a project you are using every day :) !

Selecting a Projects and Issue

The best project is a project you are using every day :) !

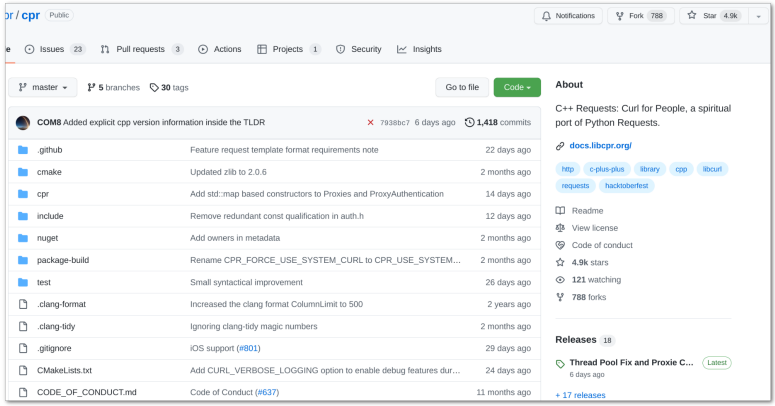
Some other great ways to discover new projects:

- Awesome First PR Opportunities
- I want to get involved!
- good first issue
- GitHub Explore
- Contrib

Checking if the Project is Still Alive

Some great ways to discover new projects:

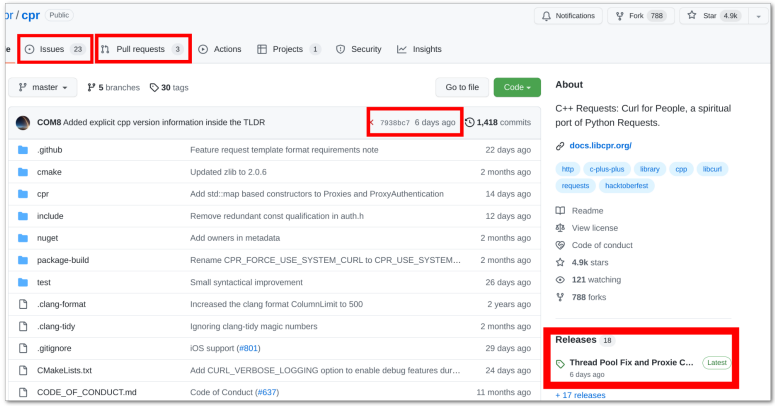
- Are there a lot of open Issues and especially unmerged Pull Requests?
- Are Issues and Pull Requests labeled?
- How responsive are the maintainers?
- How does the maintainer react to issues and feature requests?
- When was the last commit?
- How many contributors are there?



Checking if the Project is Still Alive

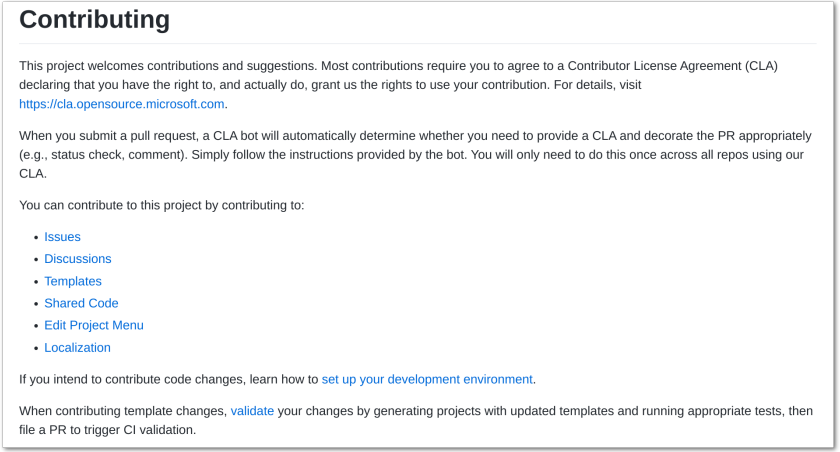
Some great ways to discover new projects:

- Are there a lot of open Issues and especially unmerged Pull Requests?
- Are Issues and Pull Requests labeled?
- How responsive are the maintainers?
- How does the maintainer react to issues and feature requests?
- When was the last commit?
- How many contributors are there?



Checking for Contribution Guidelines

Make sure, you check for contribution guidelines. Usually located inside the `CONTRIBUTING.md` file in the root of the project repository.
This can look something like this:



Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit https://cla.opensource.microsoft.com.

When you submit a pull request, a CLA bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., status check, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

You can contribute to this project by contributing to:

- Issues
- Discussions
- Templates
- Shared Code
- Edit Project Menu
- Localization

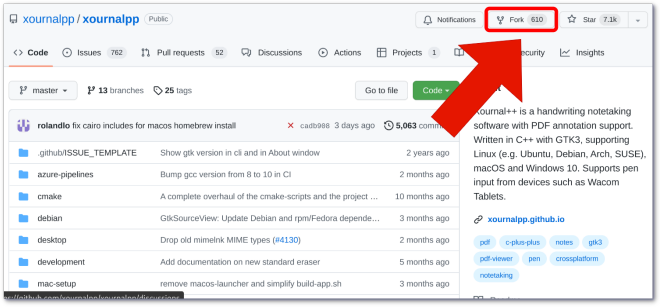
If you intend to contribute code changes, learn how to set up your development environment.

When contributing template changes, validate your changes by generating projects with updated templates and running appropriate tests, then file a PR to trigger CI validation.

Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

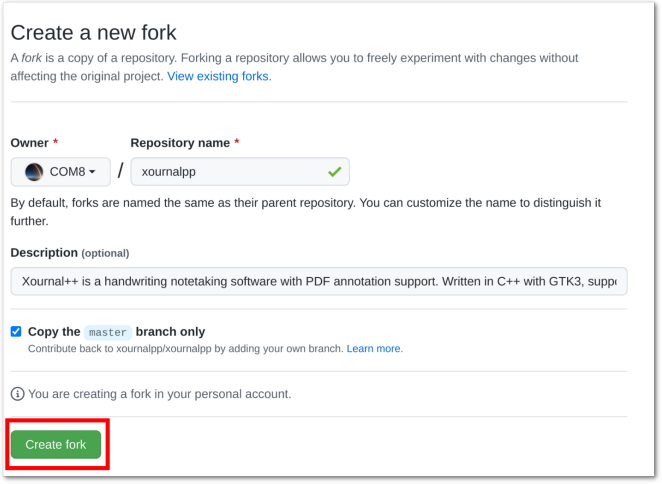
1. Fork the repository.



Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

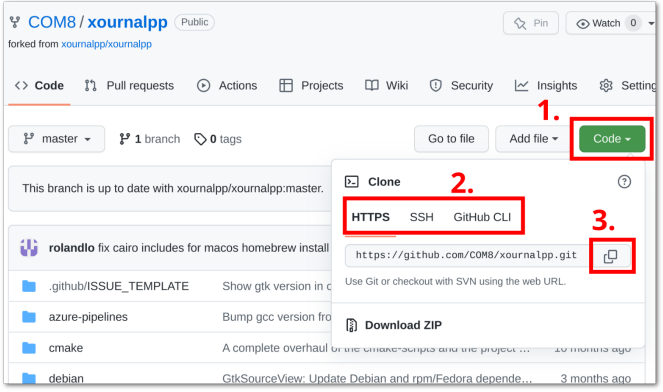
1. Fork the repository.



Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

1. Fork the repository.
2. Clone your fork.



Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

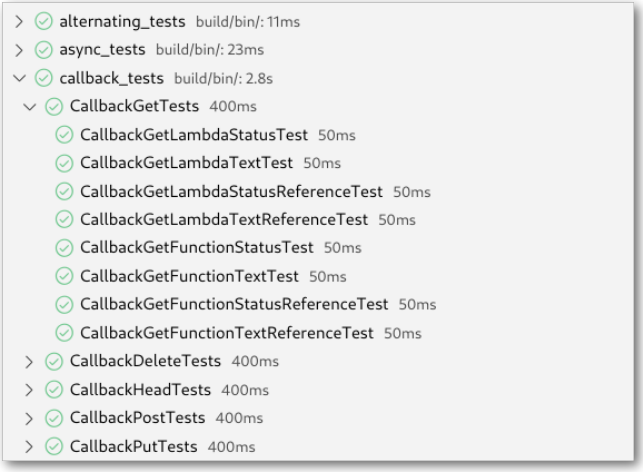
1. Fork the repository.
2. Clone your fork.
3. Get the project to compile.

```
[build] [284/286  96% :: 130.534] Linking CXX executable bin/patch_tests
[build] [285/286  96% :: 130.679] Linking CXX executable bin/structures_tests
[build] [286/286  96% :: 130.697] Linking CXX executable bin/util_tests
[build] [286/286  97% :: 130.741] Linking CXX executable bin/alternating_tests
[build] [286/286  97% :: 130.744] Linking CXX executable bin/options_tests
[build] [286/286  97% :: 130.802] Linking CXX executable bin/encoded_auth_tests
[build] [286/286  98% :: 130.887] Linking CXX executable bin/proxy_auth_tests
[build] [286/286  98% :: 130.989] Linking CXX executable bin/version_tests
[build] [286/286  98% :: 131.115] Linking CXX executable bin/download_tests
[build] [286/286  99% :: 131.166] Linking CXX executable bin/interceptor_tests
[build] [286/286  99% :: 131.203] Linking CXX executable bin/multiperform_tests
[build] [286/286 100% :: 131.248] Linking CXX executable bin/ssl_tests
[build] Build finished with exit code 0
```

Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

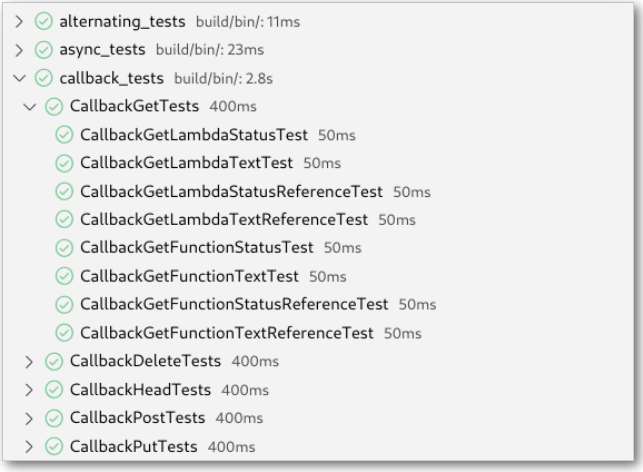
1. Fork the repository.
2. Clone your fork.
3. Get the project to compile.
4. Compile and run *all* unit tests once.



Forking and Starting to Work

So much for our theoretical project, let's get our hands dirty!

1. Fork the repository.
2. Clone your fork.
3. Get the project to compile.
4. Compile and run *all* unit tests once.
5. **You are ready to go!**



Contributing

Some guidelines for contributing to projects (issues/PRs/...):

- Not fluent in English? Use a translator like DeepL or Google Translate.
- What is the expected behavior?
- What is the actual behavior?
- What environment are you working in?
- If available, use Markdown[6] for highlighting!
- Give a bit of context and some examples like a small project or screenshots.

Further information: https://developers.google.com/blockly/guides/modify/contribute/write_a_good_issue

Take your time!
A well-written and formatted issue increases your chances for a fast reply drastically!

[6] https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf

Contributing

Some examples. . .

- https://github.com/libcpr/cpr/issues/605
- https://github.com/libcpr/cpr/issues/595
- https://github.com/libcpr/cpr/issues/627
- https://github.com/libcpr/cpr/issues/611
- https://github.com/libcpr/cpr/issues/832
- https://github.com/libcpr/cpr/issues/607

Coding Together Sessions

We offer voluntarily getting started coding sessions every **Thursday between 17:00 and 18:00** in weeks where there is no progress report, just a lecture. Starting this Thursday!
Where? Our lecture BBB room.

Goals

- Help you get started contributing to Open Source projects.
- Answer questions.
- Feel free to join if you are working on your contributions at this time anyway. We will also be coding if there are no questions ☺.