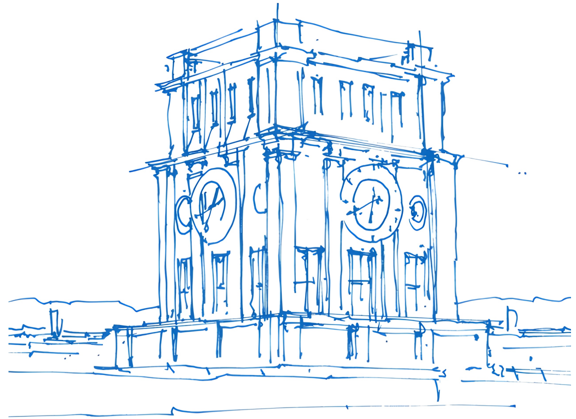
Open Source Lab

Git Advanced

Fabian Sauter, Christian Menges, Alexander Stephan

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

Garching, October 23, 2024



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 1 ”Git” by xkcd

And now let's continue. . .

(With some advanced stuff.)

Dealing with Merge Conflicts

Merge conflicts occur when git is unable to merge changes from two commits since both change the same lines in a file. They can happen when...

- merging a branch.
- rebasing a branch.
- cherry picking a commit.

Now **you** have to decide which code to keep!
Example

```
$ git merge feature-main
Auto-merging reader.cpp
CONFLICT (content): Merge conflict in reader.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

Dealing with Merge Conflicts

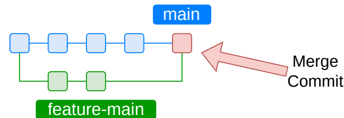
Merge conflicts occur when git is unable to merge changes from two commits since both change the same lines in a file. They can happen when. . .

- merging a branch.
- rebasing a branch.
- cherry picking a commit.

Now **you** have to decide which code to keep!

Example

```
$ git merge feature-main
Auto-merging reader.cpp
CONFLICT (content): Merge conflict in reader.cpp
Automatic merge failed; fix conflicts and then commit the result.
```



Dealing with Merge Conflicts

Git changed the `reader.cpp` file in a way, so now have to decide what to keep.

```
<<<<<<< HEAD
int main(int /*argc*/, char** /*argv*/) { }
=======
int main(void) { }
>>>>>>> feature-main
```

Once we are done, we can commit our decision.

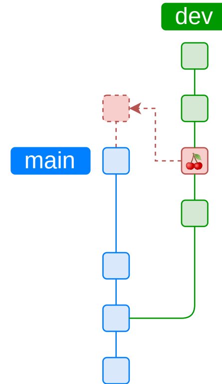
```
git commit -m "Merged feature-main into main"
```

Cherry Picking

Useful when you committed changes to the wrong branch or for backporting features to other versions.
Allows you to apply the changes from one commit to your current branch.
Use sparingly to prevent duplicate commits, which will result in merge conflicts later on.

```
git cherry-pick <commit-hash>
```

Sometimes merging or rebasing is more appropriate.



Exercise

- Check for changes on your remote "`git fetch`".
- Cherry-pick the commit "496217c630cbd69be292340da7573e2d66d3afef" (Updated the second maintainer).
- Use "`git status`" to get a list of conflicts. Resolve them.
- Continue your cherry-pick with the commands suggested to you by git.
- Push your new branch to `origin` (`git push --set-upstream origin 1.10.x_YourName`).

In case you deleted your state from last week:

```
git clone git@gitlab.lrz.de:open-source-lab/git-demo-cpr.git
cd git-demo-cpr
git checkout origin/1.10.x
git branch 1.10.x_YourName
git switch 1.10.x_YourName

nano README.md
git add README.md # Replace Fabian Sauter with your name as maintainer
git commit -m "Updated the maintainer"
```

Worktree

Motivation: It's often necessary to quickly switch between branches. This can be a bit of a hassle.

```
git stash
git switch <new_branch>
... # Possibly stashing of new changes and commits in our branch.
git switch <old_branch> # Switch back to the previous branch.
git stash pop
```

`git worktree` helps manage multiple worktrees and therefore simplifies the workflow when constantly switching branches.

Worktree

Motivation: It's often necessary to quickly switch between branches. This can be a bit of a hassle.

```
git stash
git switch <new_branch>
... # Possibly stashing of new changes and commits in our branch.
git switch <old_branch> # Switch back to the previous branch.
git stash pop
```

`git worktree` helps manage multiple worktrees and therefore simplifies the workflow when constantly switching branches.

You can start from an existing repository and create/add a new worktree in a specified directory.

```
# Creates a new directory that contains the <branch>.
git worktree add <directory> <branch>
# List all worktrees for the current repository.
git worktree list
```

Worktree

Motivation: It's often necessary to quickly switch between branches. This can be a bit of a hassle.

```
git stash
git switch <new_branch>
... # Possibly stashing of new changes and commits in our branch.
git switch <old_branch> # Switch back to the previous branch.
git stash pop
```

`git worktree` helps manage multiple worktrees and therefore simplifies the workflow when constantly switching branches.

You can start from an existing repository and create/add a new worktree in a specified directory.

```
# Creates a new directory that contains the <branch>.
git worktree add <directory> <branch>
# List all worktrees for the current repository.
git worktree list
```

Alternatively, you can start from a bare repository to avoid the initial checked-out working tree.

```
git clone --bare <repo> <directory>
```

Bisect

Problem: In large projects, it can be really difficult to find a commit that introduced a bug.

⇒ Use `git bisect` to find the commit that introduced the bug via a binary search.

Example

```
git bisect start # Start the bisect session.
git bisect good <commit> # Mark a commit as good.
git bisect bad <commit> # Mark a commit as bad.
git bisect bad/good <commit> # Continue marking commits until search terminates.
...
git bisect reset # End bisect session and reset branch.
```

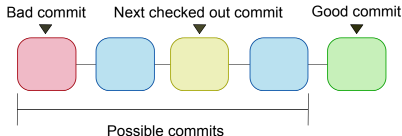
Bisect

Problem: In large projects, it can be really difficult to find a commit that introduced a bug.

⇒ Use `git bisect` to find the commit that introduced the bug via a binary search.

Example

```
git bisect start # Start the bisect session.
git bisect good <commit> # Mark a commit as good.
git bisect bad <commit> # Mark a commit as bad.
git bisect bad/good <commit> # Continue marking commits until search terminates.
...
git bisect reset # End bisect session and reset branch.
```



Exercise

- Clone: https://gitlab.lrz.de/open-source-lab/bisect-example
- Use "`git-bisect`" to find the commit, that broke the HTML formatting.
- You know the first (oldest) commit is good and the last (newest) is bad.
- Use "`git checkout <hash>`" to switch between commits.

Creating and Merging a PR Demo

(With some `git` best practices.)

Exercise

- In GitLab create a PR from your branch and resolve all issues by rebasing onto the latest `1.10.x` branch.