Git gud

Talk on Git and GitHub

Thomas Pötzsch

26.10.17

Where you should end up after this talk

- You have an idea of the inner workings and the benefits of Git and version control systems

- You can use Git

- You can use GitHub

Git

What is Git?

- Git is a file system
- Git provides a VCS

What is a VCS?

- A VCS is a Version Control System
- A VCS lets you manage different versions of files
- It keeps track of the changes made on files and documents
- A VCS provides automated workflows to reduce your workload

Reasons to use a Version Control System

- Instead of keeping track manually, a system lets you manage versions of your code

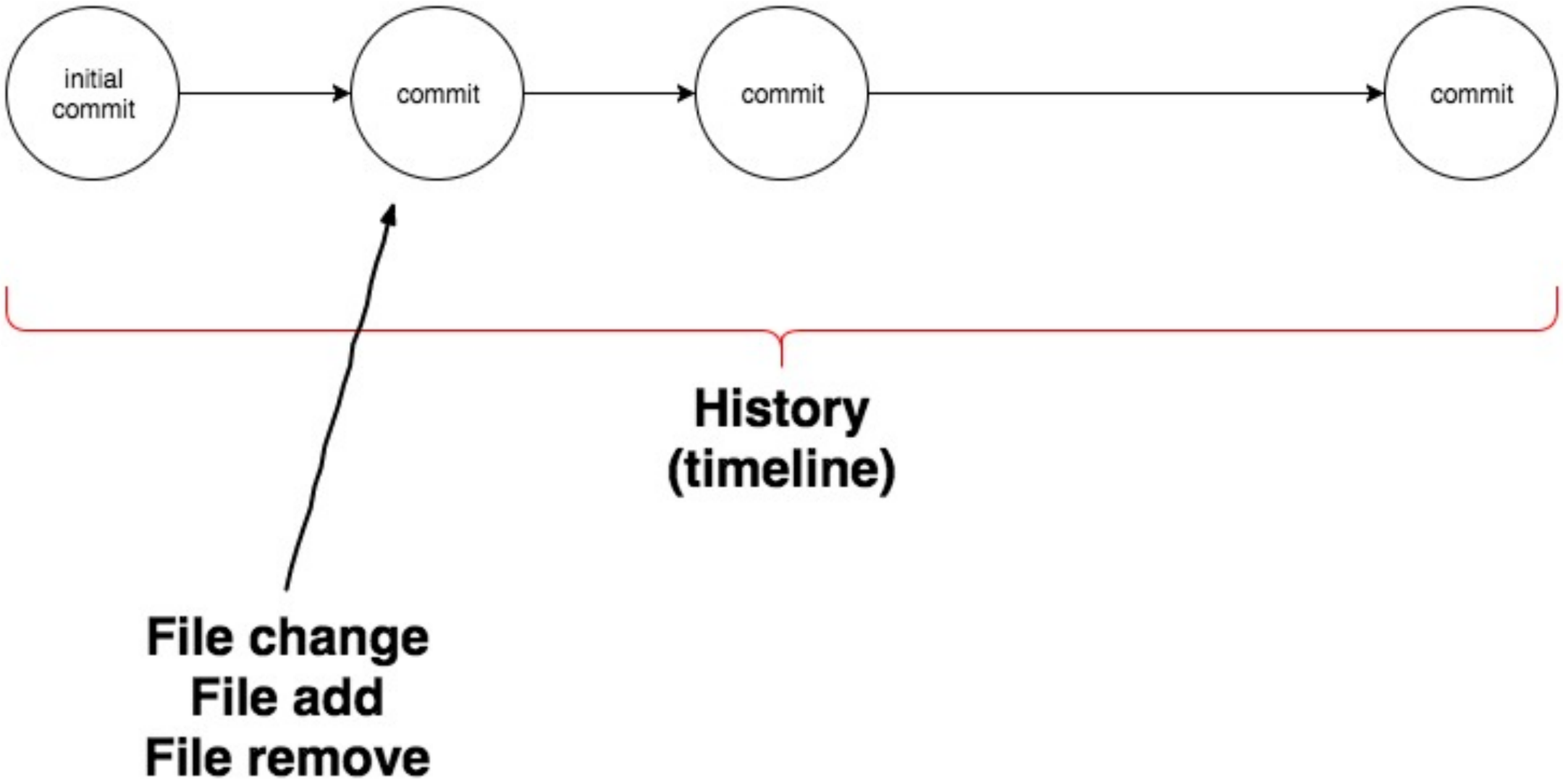
- This basically means that you can
  - Access a version history
  - Revert changes
  - Compare versions

  - Work with multiple people on the same file

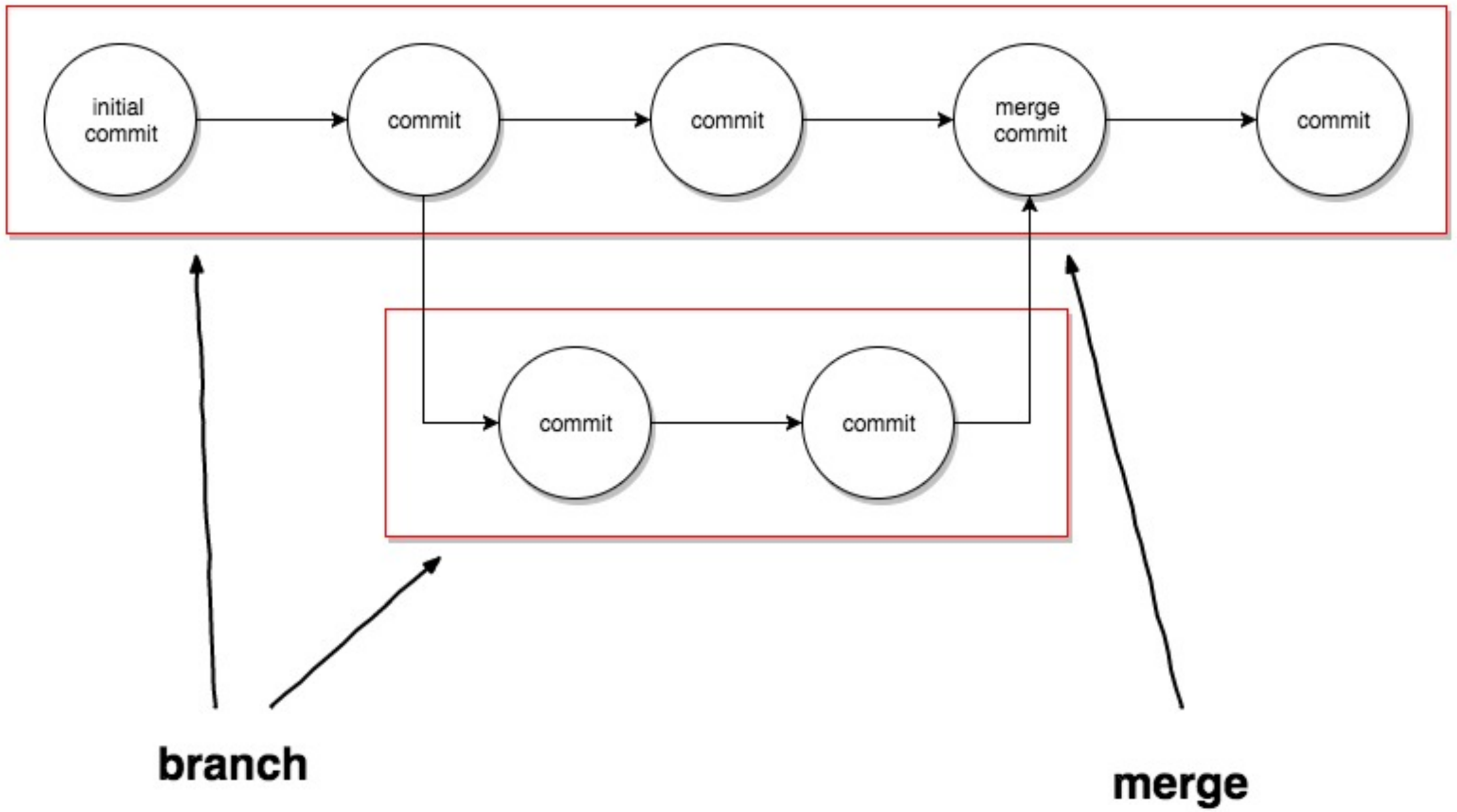
Git history

- Developed in 2005 by Linus Torvalds
- Replaced BitKeeper as a VCS in the Linux development
- Focus laid on:
  - BitKeeper-like Workflow
  - Speed
  - Security (no corruptions possible)
  - Ability to handle huge projects
  - Simplicity
- Git is still being developed (current version ^2.14.*)

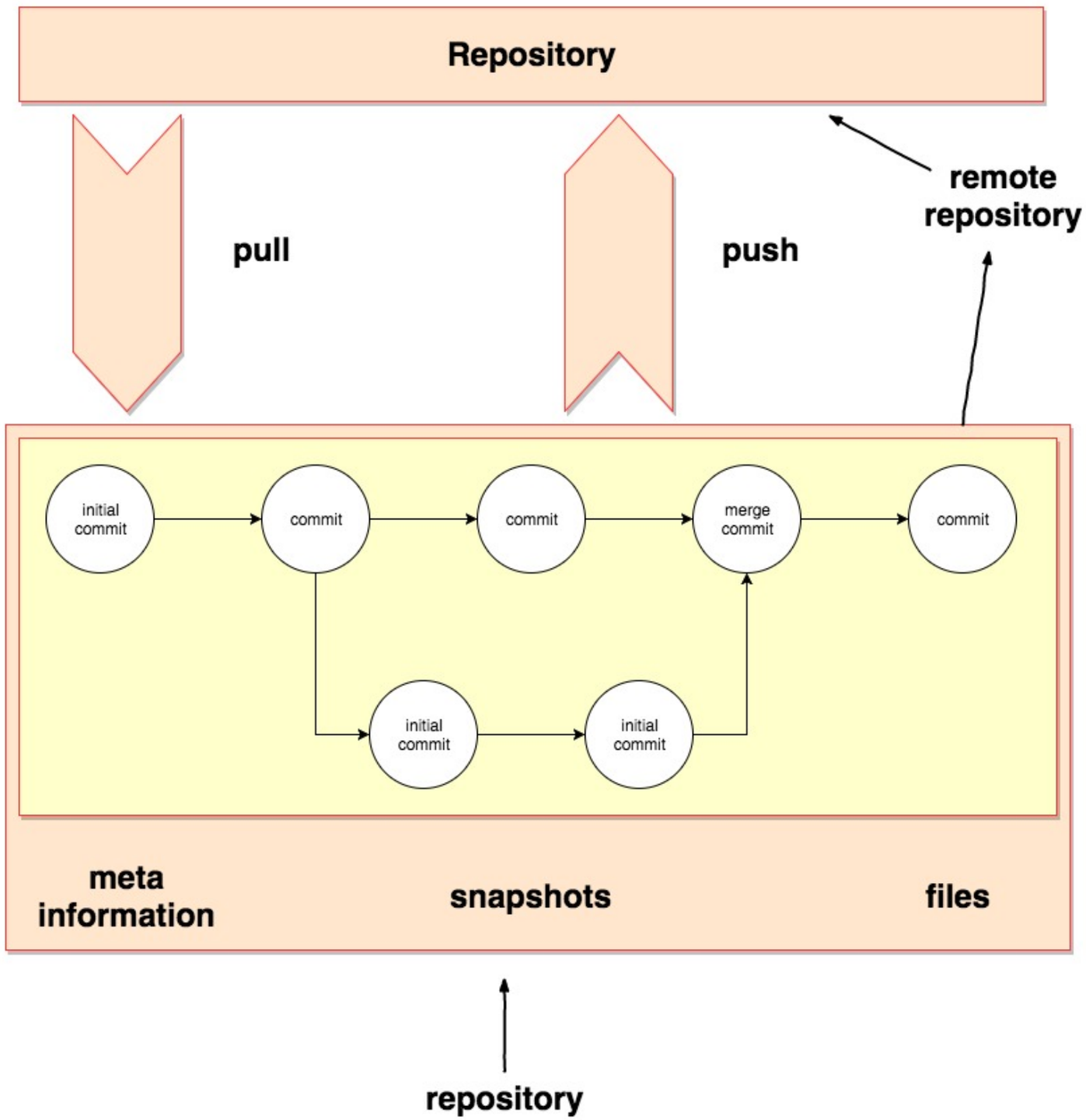
Terminology



Terminology



Terminology



How does Git work?

- Git is a content-addressable file system
- After you initiated a repository, you will see:

```
~/Workspace/Studium/SoftwareEngineering cd gitExample
~/Workspace/Studium/SoftwareEngineering/gitExample git init
Initialized empty Git repository in /Users/ry77/Workspace/Studium/SoftwareEngineering/gitExample/.git/
~/Workspace/Studium/SoftwareEngineering/gitExample  master cd .git
~/Workspace/Studium/SoftwareEngineering/gitExample/.git  master ls -la
total 24
drwxr-xr-x   9 ry77  staff  306 24 Okt 22:26 .
drwxr-xr-x   3 ry77  staff  102 24 Okt 22:26 ..
-rw-r--r--   1 ry77  staff   23 24 Okt 22:26 HEAD
-rw-r--r--   1 ry77  staff  137 24 Okt 22:26 config
-rw-r--r--   1 ry77  staff   73 24 Okt 22:26 description
drwxr-xr-x  12 ry77  staff  408 24 Okt 22:26 hooks
drwxr-xr-x   3 ry77  staff  102 24 Okt 22:26 info
drwxr-xr-x   4 ry77  staff  136 24 Okt 22:26 objects
drwxr-xr-x   4 ry77  staff  136 24 Okt 22:26 refs
```

.git folder contents

```
~/Workspace/Studium/SoftwareEngineering/gitExample/.git  master  ls -la
total 32
drwxr-xr-x  10 ry77  staff  340 24 Okt 22:46 .
drwxr-xr-x   4 ry77  staff  136 24 Okt 22:46 ..
-rw-r--r--   1 ry77  staff   23 24 Okt 22:26 HEAD
-rw-r--r--   1 ry77  staff  137 24 Okt 22:26 config
-rw-r--r--   1 ry77  staff   73 24 Okt 22:26 description
drwxr-xr-x  12 ry77  staff  408 24 Okt 22:26 hooks
-rw-r--r--   1 ry77  staff  104 24 Okt 22:46 index
drwxr-xr-x   3 ry77  staff  102 24 Okt 22:26 info
drwxr-xr-x   5 ry77  staff  170 24 Okt 22:46 objects
drwxr-xr-x   4 ry77  staff  136 24 Okt 22:26 refs
```

Current branch

Staging information

Content
(object database)

Pointers into
commit objects

Git objects

- There are three types of objects:
  - blob
  - tree
  - commit object

Blob

- Everything in git is identified by it’s SHA-1 cheksum
- Git hashes the contents of a file plus a header using SHA-1
- Git stores the object, first two chars of the hash as a directory name, last 38 chars as the file name
- The content is retrievable by addressing it via it’s SHA-1 checksum
- Problem: No file names are stored! -> only addressable via SHA-1!

Blob

```
~/gitExample git init
Initialized empty Git repository in /Users/ry77/gitExample/.git/
~/gitExample  master vim hello.txt
~/gitExample  master ls -la .git/objects
total 0
drwxr-xr-x  4 ry77  staff  136 24 Okt 22:56 .
drwxr-xr-x  9 ry77  staff  306 24 Okt 22:56 ..
drwxr-xr-x  2 ry77  staff   68 24 Okt 22:56 info
drwxr-xr-x  2 ry77  staff   68 24 Okt 22:56 pack
~/gitExample  master git add hello.txt
~/gitExample  master + cat .git/objects/49/5cc9fa8f9c127aaee426bcb7e09f46d82199d7
xK��OR04f�H���W(�H-JU�H�{%
```

Tree

- Tree objects are basically UNIX directory entries
- They contain of a SHA-1 pointer to a blob/tree as well as it’s
  - mode (file, executable, symlink)
  - type
  - name
- New snapshot = new tree object
- Problems: Still only a SHA-1 identifier, zero metadata on the tree (name, date etc.)

Commit object

- A commit object points to a tree object and holds meta

- A commit object contains of user-added information:
  - Tree pointer
  - Message
  - Preceded commit (can be none)

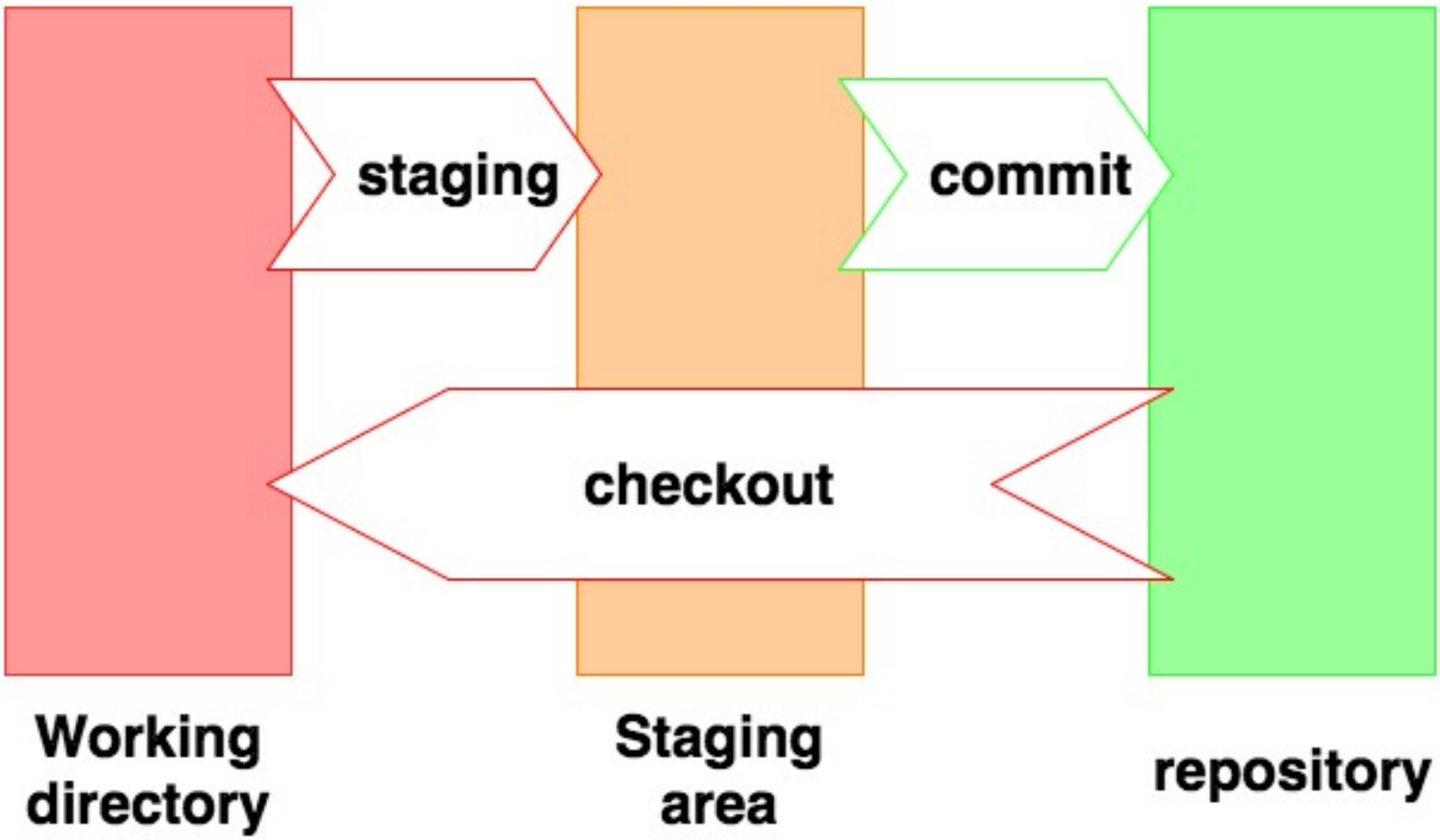
  and automatically added information
  - username, user e-mail
  - timestamp

- -> Now we have a history!

Viewing Git objects

CLI example

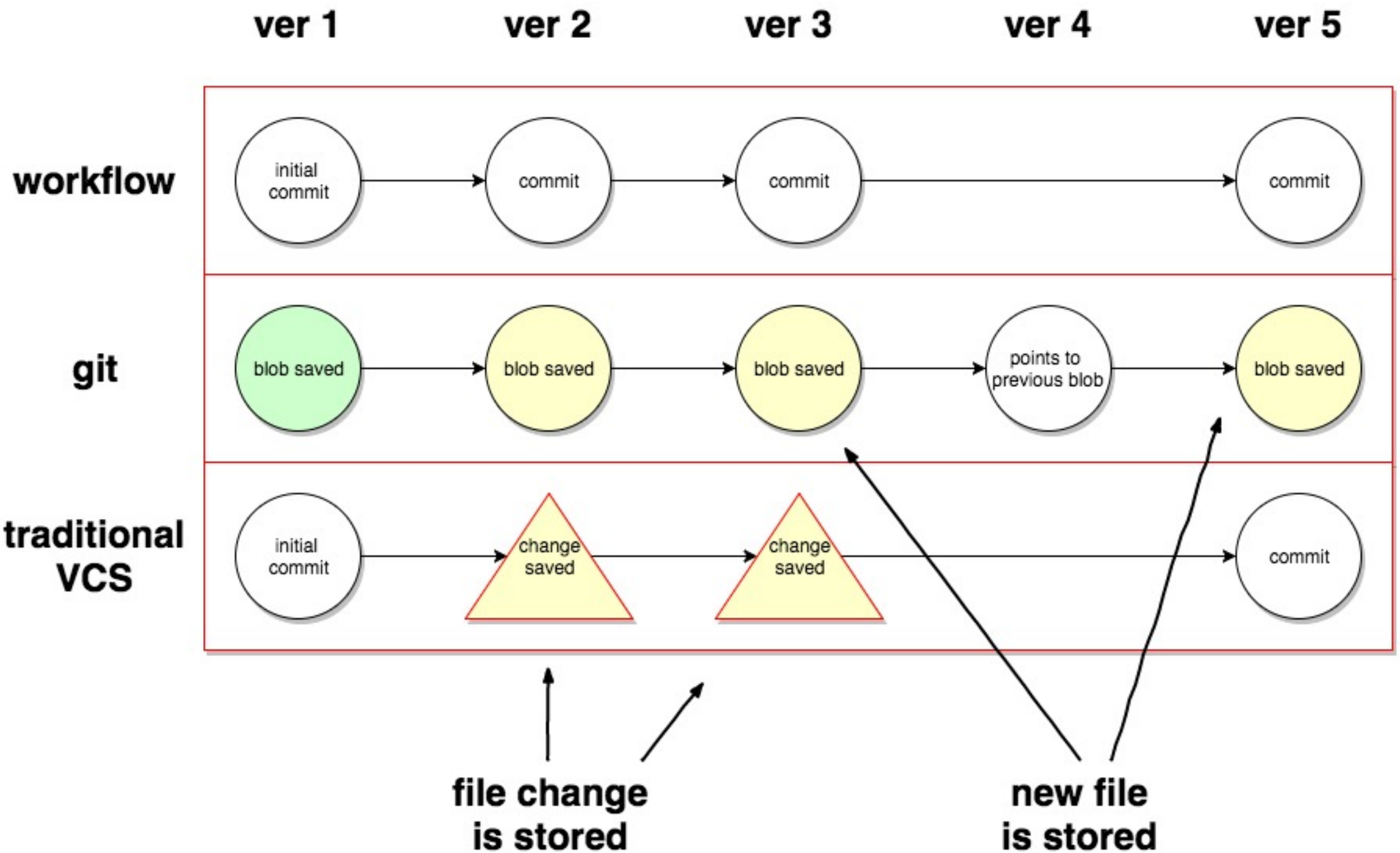
Git areas


staging
commit
checkout
Working directory
Staging area
repository

Git as a VCS

- Git is basically a file system - but what does that mean?
- Traditional VCSs store file-based changes
- Git stores snapshots

Git as a VCS


ver 1
ver 2
ver 3
ver 4
ver 5
workflow
initial commit
commit
commit
commit
git
blob saved
blob saved
blob saved
points to previous blob
blob saved
traditional VCS
initial commit
change saved
change saved
commit
file change is stored
new file is stored

Hidden workflows

- You only use user-friendly *porcelain* commands
- There are commands for using Git's inner workings

Hidden workflows

CLI example

How to use a CLI

- Enter your working directory

  ```
  cd /home/user/project
  ```

- Enter your commands (use programs)

  ```
  vim index.html
  ```

- Use parameters to change what the commands do

  ```
  ls -latr
  ```

How to use a CLI

- Get help by accessing the man pages or the help

  ```
  man git
  ```

  ```
  git help
  ```

- Watch out while using CLIs
  - They do what you want them to do
  - They won’t always warn you or forbid commands, watch out while forcing commands

```
rm -rf /
```

Basic usage

- Create a new repository:

  ```
  git init
  ```

- Clone a repository

  ```
  git clone
  ```

Basic usage

- Look at the current status

  ```
  git status
  ```

- Work on the files, change them

  ```
  vim <file to work on>
  ```

- Update the index

  ```
  git add <file to add>
  ```

Basic usage

- Create a commit object

  ```
  git commit
  ```

  ```
  git commit -m „<commit message>“
  ```

- Send your changes to the remote repository

  ```
  git push
  ```

Branching

- A branch is a pointer to a commit
  (implemented as a file that contains a commit hash)

- The file *HEAD* will point to the current branch

- The *master* is just another branch, created by git init

- To set the *HEAD* to a new branch

```
git branch <branch>
git checkout <branch>
```

```
git checkout -b <branch>
```

Branching

- Once you committed, the *HEAD* and the current branch move along

- If you checkout a different branch now and commit, you will get a *divergent history*

Merging

- What to do after multiple branches exist?
- Merge another branch into the *HEAD* using

```
git merge <branch>
```

- If the history is not divergent, the branch will move along (the pointer will point to the new commit)

Merging

- What if the history is divergent?
- Git can merge your branches together, if the same part of the same file has *not* been changed in both branches
- Git will create a *merge commit*: a commit that has two ancestors

Merging

- What if the same part of the same file has been changed?

-> You will have to handle *merge conflicts*.

- Git will inform you that merge conflicts occurred during merging

- Git will modify the file so that you can fix the merge conflict

Merging

```
~/gitExample  master  git merge FixingHelloTxt
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
✘  ~/gitExample  master ●+ >M<  vim hello.txt
~/gitExample  master ●+ >M<  cat hello.txt
Hello there!
<<<<<<< HEAD
How are you?
=======
Today is a good day.
>>>>>>> FixingHelloTxt
~/gitExample  master ●+ >M<  vim hello.txt
~/gitExample  master ●+ >M<  git add hello.txt
~/gitExample  master + >M<  git commit
[master 0c69d68] Merge branch 'FixingHelloTxt'
~/gitExample  master  cat hello.txt
Today is a good day.
~/gitExample  master
```

Merging

- Fix the merge conflict

  ```
  vim <file>
  ```

- After you have fixed the merge conflict, you can update the tree by using

  ```
  git add <file>
  ```

- After that, you can try to finish the merge, if you fixed all merge conflicts

  ```
  git commit
  ```

Advanced usage

- Some interesting commands you might need while using Git as your VCS:

- If you want to stash away your changes:

```
git stash
```

If you want to apply them:

```
git stash apply
git stash pop
```

Advanced usage

- If you forgot to stage a file or you messed up the commit message:

```
git commit --amend
```

- If you want a specific commit to be part of your tree as well:

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

- If you want to revert a commit:

```
git revert <commit>
```

Advanced usage

- If you want to change past commits to squash them, rename them or else:

  `git rebase -i` -> interactive rebasing

- If you want to search for the commit that introduced a bug:

  `git bisect start`

- If you want to create a tag:

  `git tag -a <version> -m „<message>“`

Advanced usage

- Take a look at the log

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

Benefits of the way Git is implemented

- Git is completely local and decentralized (vs Subversion)
- Everybody can develop and commit offline and then spread his work online
- Git is high speed: Git being offline and the key-value database make it extremely fast
- Git has integrity: Everything is referred to by a checksum
- It is hard to lose data: Removed files are still part of older snapshots
- Branching is *very* easy and very mighty

Get the most use out of a Version Control System

+ Make use of the workflow (commit often, push often etc.)

+ Use ASCII files (not only code, documentation or blogs as well)

+ Use scripts to get automatically created change reports and statistics

+ Use your VCS to find mistakes

Best practices

- When to commit?

  -> Commit every time you want to create a snapshot of your code:

  - Commit if you finished some aspect of the ticket you are working on
  - Separate different activities during development with different commits (group changes by their effect)
  - Commit in fear of losing progress

Best practices

- If you create multiple commits during the completion of one task, mark them as

  `WIP` (Work In Progress)

- Try to keep all WIP-commits out of the master by squashing them into previous commits

- Avoid pushing WIP-commits while working on a branch with another person (getting rid of them will get a lot harder!)

Best practices

- How to name commits?
  -> Short, understandable, precise
  -> Title: 50 characters, Comment: 50-70 character per line
  -> Add a short task description (BUGFIX, TASK, etc.)
  -> Add ticket IDs or issue IDs to trigger workflows

  ```
  WIP: PPR-5 implement search
  ```

- Think of your own clever conventions!

Best practices

- What to create branches for?

  - Create a branch every time you want to fix a bug, implement something or change a file
  - Use tickets as an orientation: Create a branch for every ticket/issue you work on
  - Throw away old branches (it just gets confusing over time)

Best practices

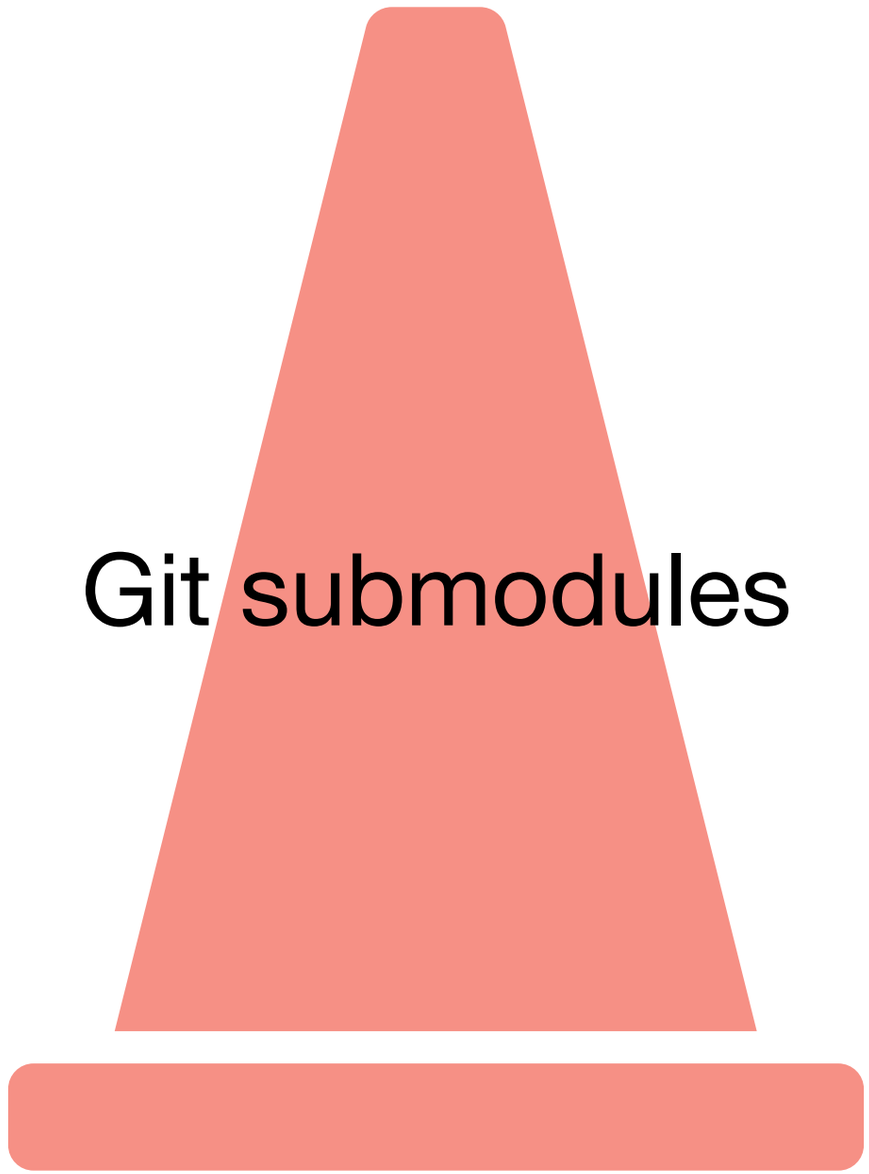
- How do I name branches?

  -> UpperCamelCase
  -> Speaking names
  -> Use ticket IDs or issue IDs
  -> Avoid more than 4 words

What to avoid

- Git is just a tool - avoid spending too much time on configuring repositories, correcting the history etc.
- Do not rely on git - it won’t fix your user-generated problems!
- Do not use Git for backups!
- Pay attention to what you are doing - Git will do whatever you tell it to do!

What to avoid at all costs



GUI tools, plugins

- Most IDEs provide a Git plugin

- The plugin will provide buttons and shortcuts for the important git features, as well as a nice GUI

- There are a lot of Git standalone GUI tools
  - Tower
  - SourceTree
  - GitKraken
  - GitHub Desktop

Git integrations

- Most Issue Trackers invite you to connect them to a Git repository

- They let you
  - Close tickets by using specific commit messages
  - Create custom workflows

Git integrations

- A lot of CI-tools let you connect them to repositories

- By using specific commit messages you can
  - trigger builds
  - trigger automated tests
  - trigger deployments

Tools for handling Merge Conflicts

- There are many tools that let you manage merge conflicts
  - DiffMerge
  - Kaleidoscope

GitHub

What is GitHub?

- GitHub is a repository hosting service
- GitHub is hosting Git repositories
- GitHub lays it’s focus heavily on collaborative developing

-> Comparable to GitLab

GitHub features

- Repository hosting (used for remote repositories)
- Wikis
- Issue tracking
- GitHub Pages
- Statistics, graphs on repositories and users
- GitHub Gists

GitHub features

- Pull requests
- CI
- Private repositories (premium feature)

Are there any questions?