Introduction to git

Version Control System

Why use a version control system?

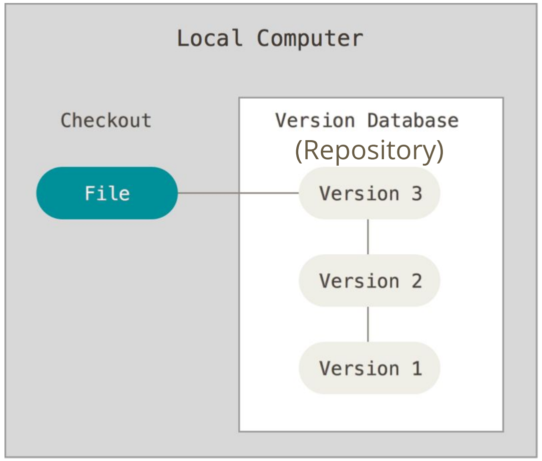
- Backup!
  - Always backup your code on an external server, all local changes will be deleted when the computer restarts
- Keep track of all the changes in the code
  - Is something wrong? Just check which lines are changed!
- Easy to experiment with new features
  - Add new features and experiment in other branches, and leave the stable code that works in the master branch

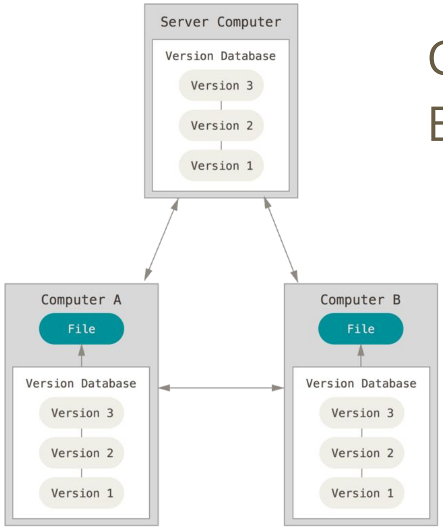
NOTE: Git and Version Control Systems is not part of the curriculum, but rather a great tool that will help you during the labs.

Distributed Version Control System

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- With a distributed VCS like git, the local repo is a full image of the version database on the server. That means each clone is its own backup, and also making it easy to switch provider. It doesn't matter if you use a local computer, a server on NTNU, or a provider like github or bitbucket. With remotes, you can actually use all at the same time!

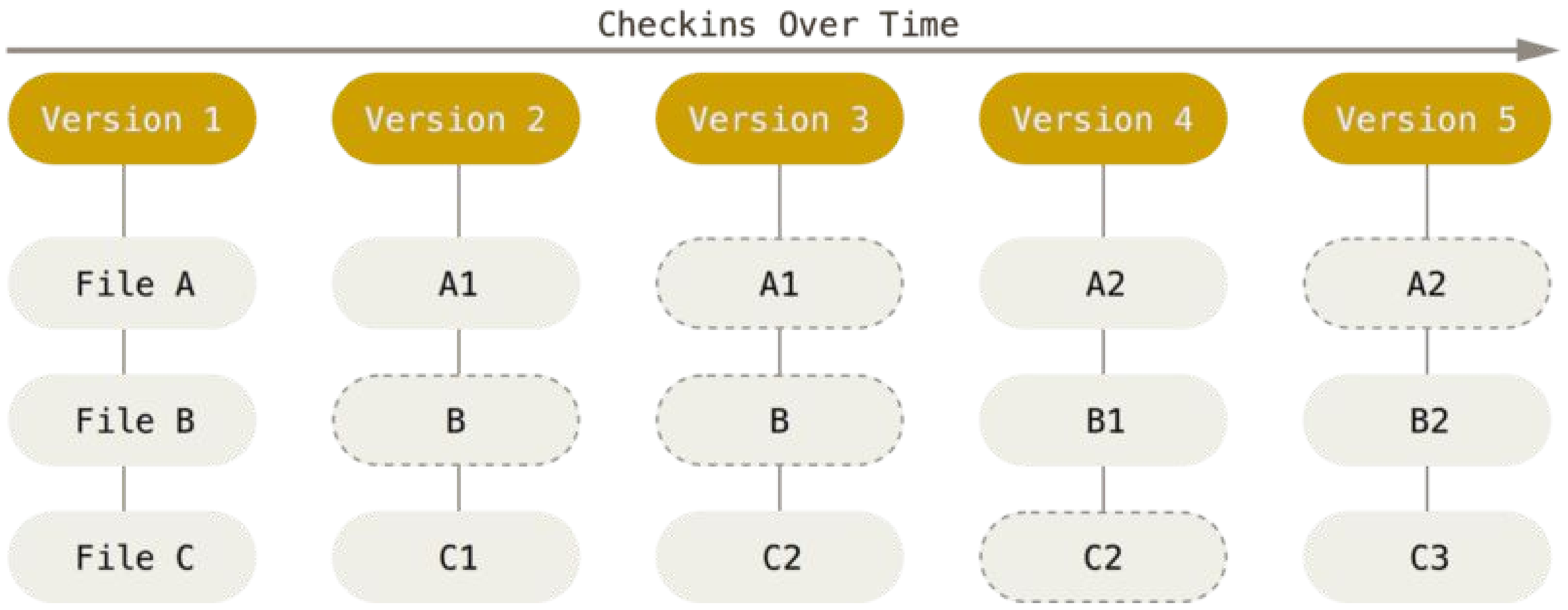




Github/
Bitbucket

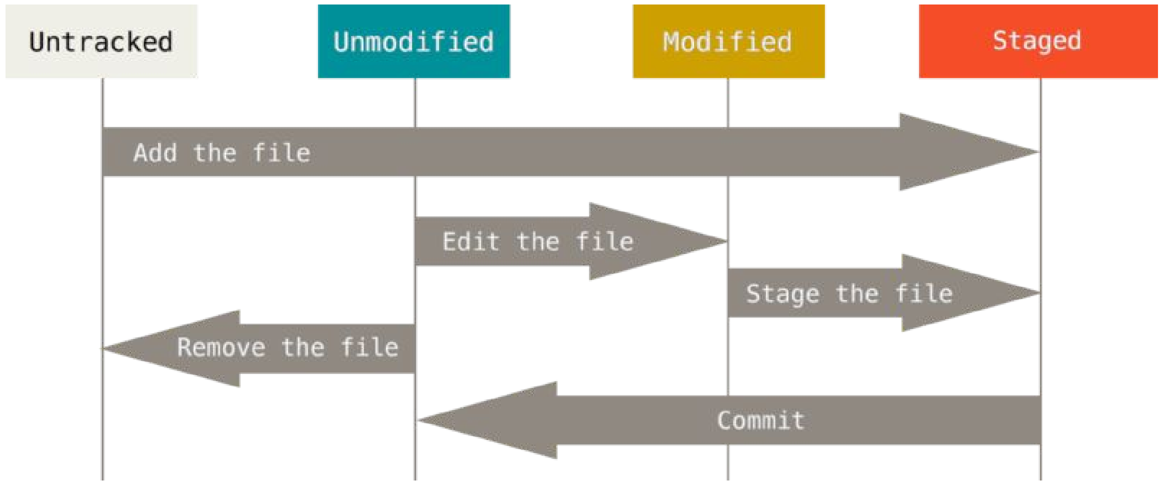
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Commit: Snapshots of files



https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

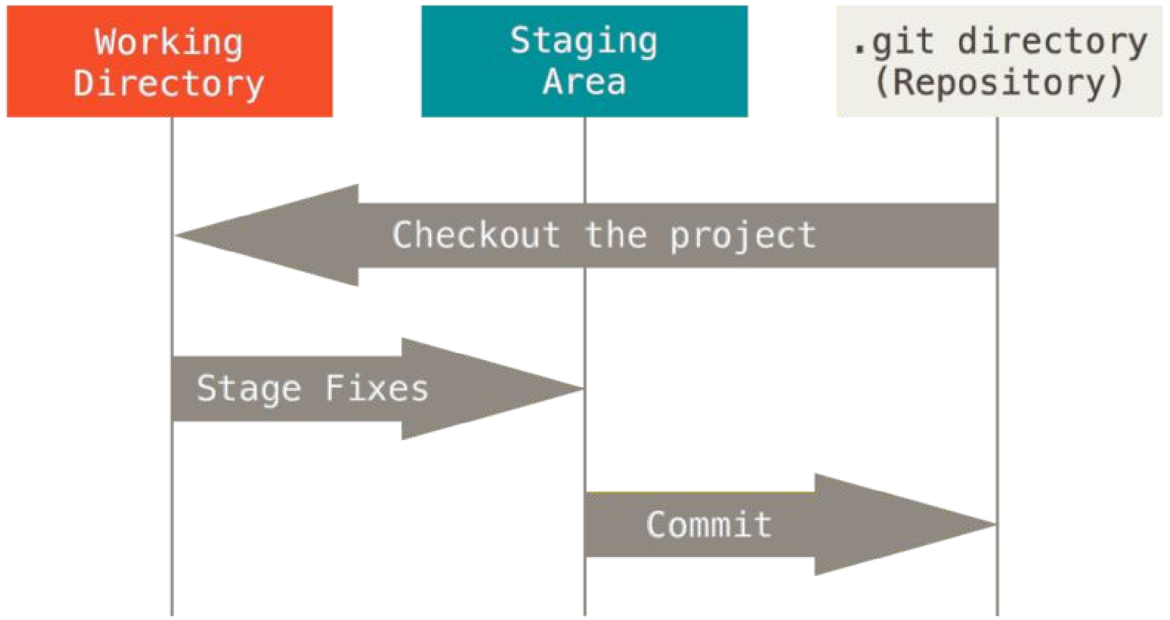
Recording changes to the repository



Untracked: Git doesn’t keep track of changes in the file, and it’s not stored in the repository.
Unmodified: Git keep track of this file, but there is not any differences between this file and the committed version in the repository.
Modified: Modified means that you have changed the file but have not committed it to your database yet.
Staged: Staged means that you have marked a modified file in its current version to go into your next commit snapshot.

https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

The three stages

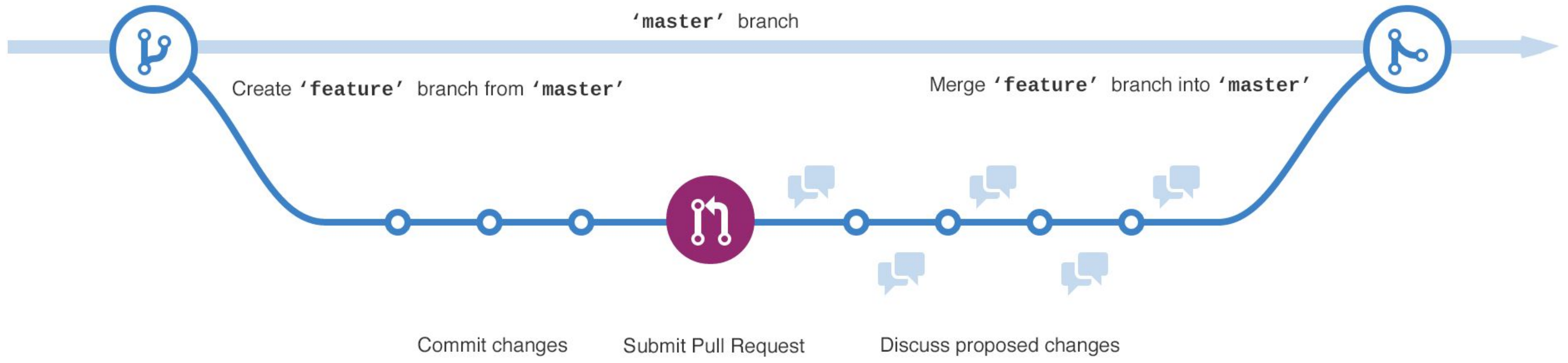


1. You modify files in your working tree.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Branches!

- This is a useful tool when developing, but you could get into trouble when merging a branch back into the master branch.
- Leave the stable code that works in the master branch, and experiment with new features in other branches. If something breaks, you can switch branches and test if it was a hardware failure, or if you created a bug in the code.


'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

https://guides.github.com/activities/hello-world/

Useful command line instructions

Configure git:

```
git config --global user.name "name"
```
Sets the name you want attached to your commit transactions

```
git config --global user.email "email address"
```
Sets the email you want attached to your commit transactions

Create local repository:

```
git clone [url]
```
Downloads a project and its entire version history

Make changes:

```
git status
```
Lists all new or modified files to be committed

```
git add [file]
```
Create snapshot of the file in preparation for versioning

```
git diff --staged
```
Shows file differences between staging and the last file version

```
git reset [file]
```
Unstages the file, but preserve its contents (opposite of git add)

```
git commit -m "descriptive message"
```
Records file snapshots permanently in version history

Review history:

```
git log
```
Lists version history for the current branch

```
git diff [branch] [branch] -- [file]
```
Shows content differences between two branches of one file

Synchronize changes with remote server:

```
git push
```
Uploads all local commits to GitHub

```
git pull
```
Downloads bookmark history and incorporates changes

Branches:

```
git branch
```
Lists all local branches in the current repository

```
git branch [branch-name]
```
Creates a new branch

```
git checkout [branch-name]
```
Switches to the specified branch and updates the working directory

```
git branch -d [branch-name]
```
Deletes the specified branch

Bash terminal basics

- On the lab we recommend that you use ‘Git bash’, a bash terminal emulator with support for Git and basic Unix commands. With this there is no difference using Git on Linux, Windows or MacOS (or any other Unix systems)
- If you are not comfortable with using terminal tools, you can use the Git GUI client instead which is also installed on the lab computers.
- There is no need for much experience working with a terminal, but it could be convenient knowing how to navigate around

Basic terminal commands:

```
cd <directory_name>
```
Change directory, navigate to this directory
```
cd ..
```
Change directory, go back one step
```
ls
```
List files in the current directory

Set up a git repo on your machine

1. **Create account on Github or Bitbucket**
2. **Create a repository for your project on Github/Bitbucket**
    - Include **.gitignore** for C-projects
    - The **.gitignore** file contains a list of files and folders that you don't want to track with git. Git will ignore all file and directory names listed in this file.
3. **Initialize git on your machine and clone/download the repository:**

```
git config --global user.name "name"
git config --global user.email "email address"
git clone [url-to-repo]
```

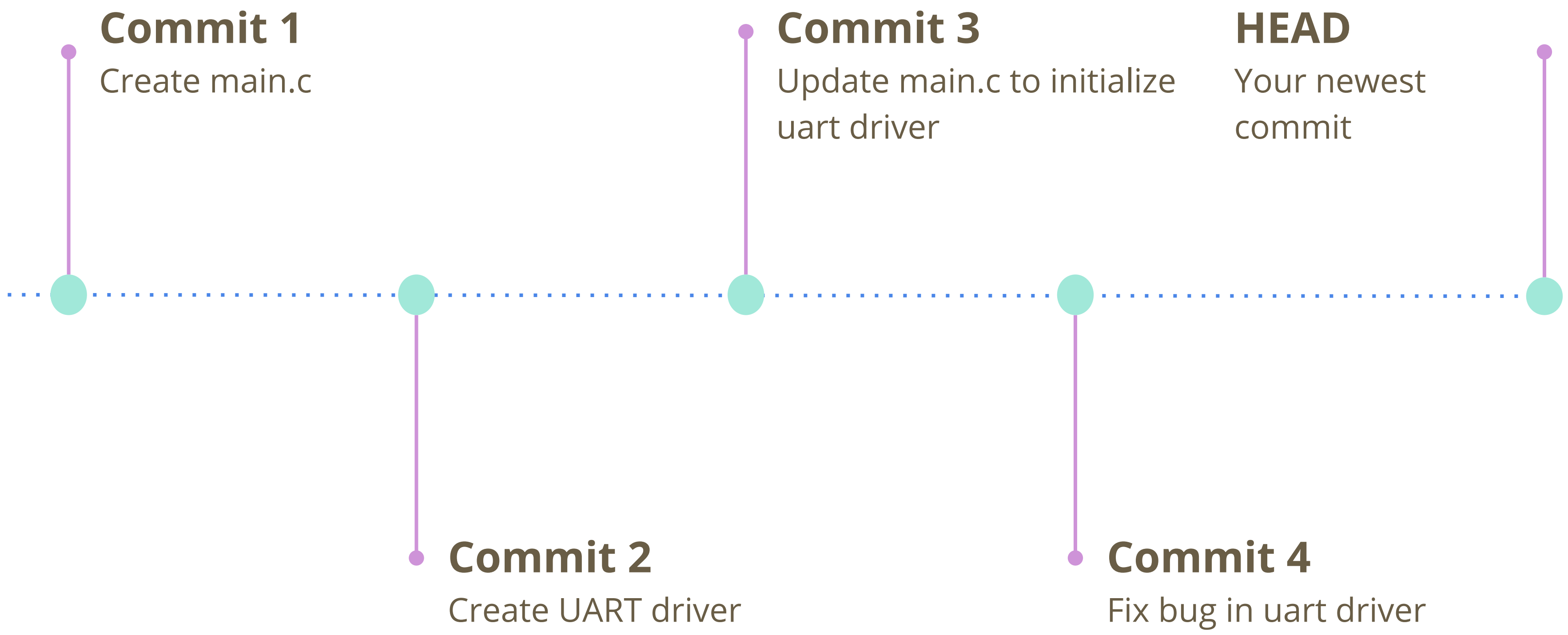
4. **Create a project in Atmel studio**
    - Modify **.gitignore** to ignore the 'Debug' folder (generated when you build your project) and all other temporary files you don't want to keep track of.

Workflow when coding:

1. **Make sure your local repo has all the latest changes by downloading the history from the server** (e.g. github)
   ```
   git pull
   ```
2. **Write code and have fun modifying files**
3. **Stage the modified files**
   ```
   git status             (see which files are changed)
   git add [file1]        (stage a file)
   git add [file2]
   ...
   ```
4. **Commit the new changes** (take the snapshot to create a new version)
   ```
   git commit -m "descriptive commit message"
   ```
   or

   `git commit` (this will open a vim terminal editor that let you write a longer commit message, use with care...)
5. **Repeat step 2, 3 and 4 until you are done for the day.**
   Try to commit every time you make a logical change in a file.
6. **Before you leave, push all your changes onto a remote server to backup your code.**
   (to avoid forgetting, you can might aswell do this every time with step 4)
   ```
   git push
   ```

Example of a commit history

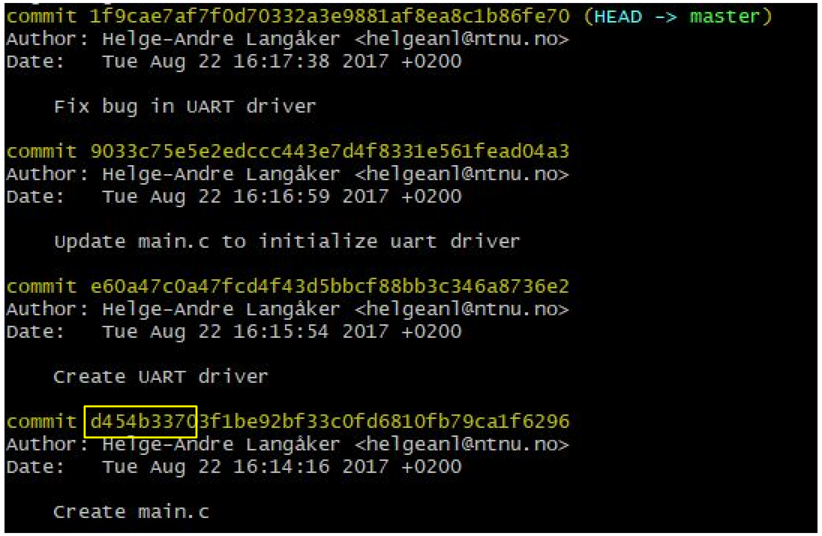


Example: Roll back to a previous commit

Something has gone wrong, and you want to try out an older version of your code?

1. **Find the commit hash for the version you want**
   `git log`
2. **Use checkout to roll back to commit “Create main.c”**
   `git checkout d45b3370`
3. **Step 2 leaves you in a ‘detached HEAD’ state, meaning you are not in a branch. Avoid doing changes in this state, and go back to your previous branch when you are done testing.**
   `git checkout master`

Output from ‘git log’



```
commit 1f9cae7af7f0d70332a3e9881af8ea8c1b86fe70 (HEAD -> master)
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:17:38 2017 +0200

    Fix bug in UART driver

commit 9033c75e5e2edccc443e7d4f8331e561fead04a3
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:16:59 2017 +0200

    Update main.c to initialize uart driver

commit e60a47c0a47fcd4f43d5bbcf88bb3c346a8736e2
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:15:54 2017 +0200

    Create UART driver

commit d454b33703f1be92bf33c0fd6810fb79ca1f6296
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:14:16 2017 +0200

    Create main.c
```

Example: Roll back a single file to a previous commit

In some cases you may want to only check out an older version of a single file.

1. **Find the commit hash for the version you want**

   `git log`

2. **Use checkout to roll back to commit “Create main.c”**

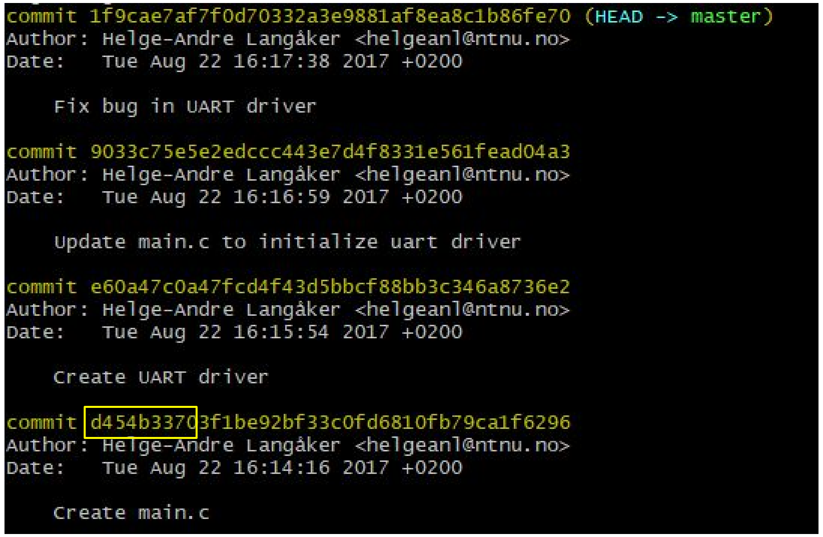
   `git checkout d45b3370 -- main.c`

3. **When you checkout only a single file, you do not leave your current branch.** ‘main.c’ **is now a modified file compared to the HEAD (latest commit) of your current branch.**
   - **Make a new commit with these changes**

     ```
     git add main.c
     git commit -m “Bugfix main.c”
     ```
   - **OR reset the rollback when you are done testing**

     ```
     git reset HEAD -- main.c
     git checkout -- main.c
     ```

Output from ‘git log’



```
commit 1f9cae7af7f0d70332a3e9881af8ea8c1b86fe70 (HEAD -> master)
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:17:38 2017 +0200

    Fix bug in UART driver

commit 9033c75e5e2edccc443e7d4f8331e561fead04a3
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:16:59 2017 +0200

    Update main.c to initialize uart driver

commit e60a47c0a47fcd4f43d5bbcf88bb3c346a8736e2
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:15:54 2017 +0200

    Create UART driver

commit d454b33703f1be92bf33c0fd6810fb79ca1f6296
Author: Helge-Andre Langåker <helgeanl@ntnu.no>
Date:   Tue Aug 22 16:14:16 2017 +0200

    Create main.c
```

Take one step at a time

- Git has a lot of features to offer, and can therefore be intimidating to beginners if you try to learn all at once in the beginning.
- It is a great tool to make you more efficient, and give you better control of your code, but **don’t** use too much time on the lab learning Git. Take a tour through a Git tutorial to get the basics and learn while doing one step at a time. (time is a precious resource on the lab!)

Some useful links

Create repository

- https://github.com
- https://bitbucket.org/

Git tutorials

- https://guides.github.com/activities/hello-world/
- https://try.github.io
- https://github.com/git-game/git-game

Git documentation

- https://git-scm.com/book/en/v2
- https://git-scm.com/docs

Git cheat sheet

- http://ndpsoftware.com/git-cheatsheet.html
- https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf