

Installation - GUI

¹Graphical User Interface
²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:

¹Graphical User Interface

²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)

¹Graphical User Interface

²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)
 - Github Desktop (only for Mac & Windows)

¹Graphical User Interface

²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)
 - Github Desktop (only for Mac & Windows)
 - gitg (only for Linux & Windows)

¹Graphical User Interface

²Command Line Interface

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)
 - Github Desktop (only for Mac & Windows)
 - gitg (only for Linux & Windows)
 - See <https://git-scm.com/downloads/guis> for more

¹Graphical User Interface

²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)
 - Github Desktop (only for Mac & Windows)
 - gitg (only for Linux & Windows)
 - See <https://git-scm.com/downloads/guis> for more
- Most IDEs have a Git GUI built-in

¹Graphical User Interface

²Command Line Interface

Installation - GUI

Some options:

- Pure Git GUI¹ clients:
 - git-gui & gitk (comes with the Git CLI)
 - Github Desktop (only for Mac & Windows)
 - gitg (only for Linux & Windows)
 - See <https://git-scm.com/downloads/guis> for more
- Most IDEs have a Git GUI built-in

Nice and all, but we will use the CLI².

¹Graphical User Interface

²Command Line Interface

Installation - CLI

Installation - CLI

- Windows

- Using *winget*:

```
winget install --id Git.Git -e --source winget
```

- Using an installer or portable version:

<https://git-scm.com/downloads/win>

Installation - CLI

- Windows

- Using *winget*:

```
winget install --id Git.Git -e --source winget
```

- Using an installer or portable version:

<https://git-scm.com/downloads/win>

- Linux

- You already have it (-:
- Through your package manager:

- `apt install git`
- `dnf install git`
- `pacman -S git`

Installation - CLI

• Windows

- Using *winget*:

```
winget install --id Git.Git -e --source winget
```

- Using an installer or portable version:

<https://git-scm.com/downloads/win>

• Linux

- You already have it (-:
- Through your package manager:

- `apt install git`
- `dnf install git`
- `pacman -S git`

• MacOS

- Using Homebrew:
 - `brew install git`
- Install Xcode, that ships with Git

Install Git!

Get out your laptop and install Git on it if you haven't already, we will start using it directly after the break

Installation successful?

Installation successful?

```
johanv@my-machine: ~$ git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
  [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
  [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
  [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
  [--super-prefix=<path>] [--config-env=<name>=<envvar>]
  <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: `git help tutorial`)

```
clone      Clone a repository into a new directory
init       Create an empty Git repository or reinitialize an existing one
```

work on the current change (see also: `git help everyday`)

```
add        Add file contents to the index
mv         Move or rename a file, a directory, or a symlink
restore    Restore working tree files
rm         Remove files from the working tree and from the index
```

examine the history and state (see also: `git help revisions`)

```
bisect     Use binary search to find the commit that introduced a bug
diff       Show changes between commits, commit and working tree, etc
grep       Print lines matching a pattern
log        Show commit logs
```

Configuration

Why:

- Let git know who's editing.
- Let git know how to do some things.
- Usually only change after setup.

How:

- Using commands:

```
git config --global core.editor 'nano'
```

```
git config --global <setting> <value>
```

What:

Setting	Advised value
core.editor	nano / <preferred editor>
user.name	<your name>
user.email	<your email address>
user.useConfigOnly	true
init.defaultBranch	main
pull.ff	true

Configuration - File structure

Read `git help config` or `man git config` for all configuration options.

You can also edit using an editor:

```
git config --global --edit:
```

```
[core]
  editor = nano
[init]
  defaultBranch = main
[user]
  useConfigOnly = true
  name = Alice
  email = alice@example.com
[pull]
  ff = true
```

Tutorial

Starting a new project

Using Git Bash or other terminal we start by creating a regular folder to hold our project. Then setup the repository by doing an init.

```
$ mkdir my-summary  
$ cd my-summary  
$ git init
```

README.md

Let's start our project with a file. We will create the file *README.md*. This is a markdown type file. Markdown is a simple filetype that allows for formatted text. It is likely you already know quite a bit of markdown as it is also used in for instance Discord and (to a lesser extend) in WhatsApp.

The *README.md* file is special. For developers this is often an entry point to understand what the repository is about. On many websites such as Gitlab the *README.md* file will also be shown (formatted) on the front page of your repository website.

Your first file

Let's add some content to the *README.md* file:

```
# Git Course Summary
```

```
This _README.md_ file will contain some markdown  
text detailing what I learned during the Git  
Course.
```

We can now save the file. Note that at this point the file is only changed in your *workspace*.

Your first commit

- We move the created file to the index:
`$ git add README.md`
- And we do our first commit!
`$ git commit -m 'Initial commit'`

Your first commit



main

- We move the created file to the index:
`$ git add README.md`
- And we do our first commit!
`$ git commit -m 'Initial commit'`
- On the left you see the commit tree. Currently we are on the first commit inside the main branch.
- You can also see your first commit:
`$ git log`
`$ git log --pretty=oneline`

Changing something

Let's add some text to the *README.md* file:

```
[...]  
## How to commit  
1. Add, delete or change files  
2. You can stage your files by doing `git add  
filename` or stage everything with `git add -A`  
3. Commit using `git commit -m 'message'`, be  
sure to provide a good description of the  
changes.
```

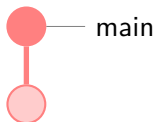
We can now save the file. Note that at this point again file is only changed in your *workspace*. Try to add it to the index and then make a second commit.

Your second commit

 — main

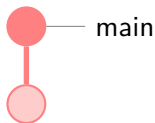
- We can move all files onto the index by using:
`$ git add -A`
- And perform the second commit:
`$ git commit -m 'Add the section: how to commit'`

Your second commit



- We can move all files onto the index by using:
`$ git add -A`
- And perform the second commit:
`$ git commit -m 'Add the section: how to commit'`
- If everything went well we now have the tree on the left. The reference HEAD is now pointing to the second commit.

Branching out



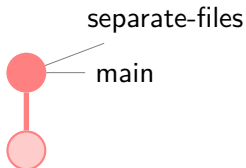
It is useful to develop larger features in a separate branch of your repository. This helps working together as well as giving the ability to switch between versions of your codebase.

Let's say we want to move our 'How to commit' section to a different file. We start by switching to a new branch:

(the `-c` argument stands for create)

```
$ git switch -c separate-files
```

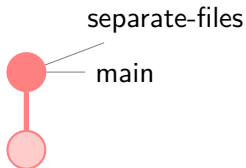
A new branch



The create command bases the new branch on the branch you were in. So at this moment the *separate-files* branch and the *main* branch are identical. We did switch to the *separate-files* branch, this is also visible in the terminal. Any new commits will be pushed to the current branch.

Let's give that a try!

Separating the files

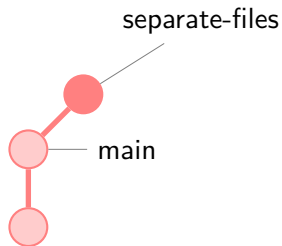


- Make a new file *TUTORIAL.md* copy the *How to commit* section over from *README.md*.
- Delete that part in the *README.md* file.
- Commit all changes:

```
$ git add -A'
```

```
$ git commit -m 'Move "How to commit" to TUTORIAL.md'
```

Separating the files

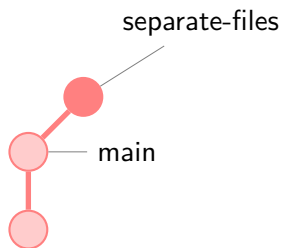


- Make a new file *TUTORIAL.md* copy the *How to commit* section over from *README.md*.
- Delete that part in the *README.md* file.
- Commit all changes:

```
$ git add -A'
```

```
$ git commit -m 'Move "How to commit" to TUTORIAL.md'
```
- The *separate-files* branch now has an extra commit on it and is no longer the same as the *main* branch.

More commits

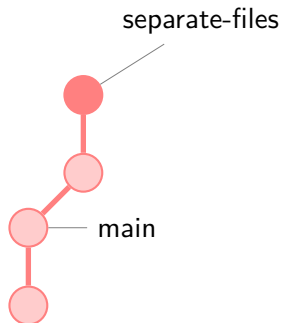


- Add a small header in *TUTORIAL.md* explaining what is in the file.
- Commit this change:

```
$ git commit -am 'Add file info for TUTORIAL.md'
```

*Note: the -a option on git commit automatically adds all changed files to the index, but it does **not** track new files!*

More commits



- Add a small header in *TUTORIAL.md* explaining what is in the file.
- Commit this change:

```
$ git commit -am 'Add file info for TUTORIAL.md'
```

*Note: the -a option on git commit automatically adds all changed files to the index, but it does **not** track new files!*
- The branch now has 2 extra commits.
- Change and save something in the *README.md* file.
- Now we want to edit something unrelated to the separate files. We try switching back to the main branch, can you?

```
$ git switch main
```


Switching branches

You cannot switch branches when your workspace is not 'clean'. You basically have a few options at this point.

- 1 Commit the last change of your workspace to the branch.
- 2 Clean all files in your workspace to the last commit (i.e. delete the changes):

```
$ git reset --hard
```
- 3 Stash your changes (Given in intermediate course)

Switching branches

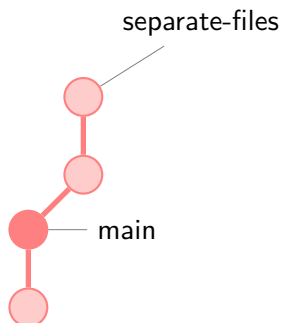
You cannot switch branches when your workspace is not 'clean'. You basically have a few options at this point.

- 1 Commit the last change of your workspace to the branch.
- 2 Clean all files in your workspace to the last commit (i.e. delete the changes):
`$ git reset --hard`
- 3 Stash your changes (Given in intermediate course)

Try the second option and switch again using:

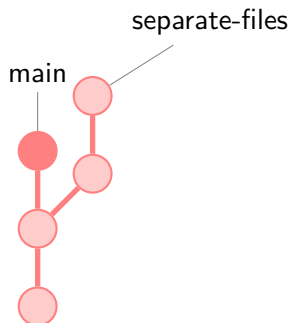
```
$ git switch main
```

Pushing to main



- Your reference is now back at the *main* branch. Notice how your files have also reverted back. *Note: Do not worry you can freely switch between the branches. As long as you don't reset or clean everything no code is ever lost when using git.*
- Let's add a commit on the main branch. Go into *README.md* and add:
`**TUTORIAL.md** - Basic tutorial on how to commit.`
- Commit changes:
`$ git commit -am 'Add file description for TUTORIAL.md'`

Pushing to main



- Your reference is now back at the `main` branch. Notice how your files have also reverted back. *Note: Do not worry you can freely switch between the branches. As long as you don't reset or clean everything no code is ever lost when using git.*
- Let's add a commit on the `main` branch. Go into `README.md` and add:
`**TUTORIAL.md** - Basic tutorial on how to commit.`
- Commit changes:

```
$ git commit -am 'Add file description for TUTORIAL.md'
```
- The `main` branch now has another commit.

Merging

Git allows us to do non-linear code editing. We can keep pushing different features to the separate branches. We can switch, compare and do all sorts of different things with this.

Eventually there comes a point where we might want to get all changes of a branch (for instance separating the *TUTORIAL.md* file) into another branch (for instance the *main* branch). We do this with merging.

Merging

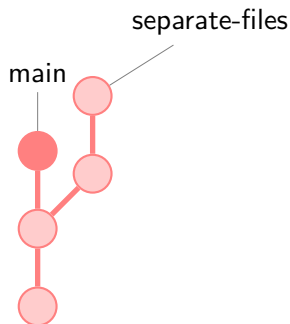
Git allows us to do non-linear code editing. We can keep pushing different features to the separate branches. We can switch, compare and do all sorts of different things with this.

Eventually there comes a point where we might want to get all changes of a branch (for instance separating the *TUTORIAL.md* file) into another branch (for instance the *main* branch). We do this with merging.

To merge the *separate-files* branch onto the current (*main*) branch perform:

```
$ git merge separate-files
```

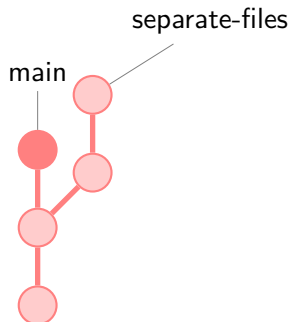
Merging - Conflict



- Merging a branch can go automatically. But not now. Since both *separate-files* and *main* have more recent changes to *README.md* we have a conflict that needs to be resolved.
- Go through *README.md* and find the conflict. With your IDE or by deleting the GIT messages you can fix your code to get the wanted result from both branches.
- When you are done fixing the *README.md* file add it to the index:

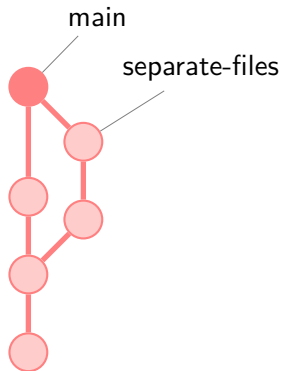
```
$ git add -A
```

Merging - Conflict



- Now continue the merge:
`$ git merge --continue`
Note: When all conflicts are resolved you will be asked to provide a message. Providing this message (by closing the file) will make a new merge commit.

Merging - Conflict



- Now continue the merge:
`$ git merge --continue`
Note: When all conflicts are resolved you will be asked to provide a message. Providing this message (by closing the file) will make a new merge commit.
- We now see a new commit on `main` and all the history of `separate-files` is also pulled in.

Working together

Now we have the basics of committing and branches down. A real power of Git is working together with other people. Git allows us to clone a remote repository and then push our commits and branches to that. Any repository can be setup as remote if there is Git server running or SSH access to it. However we almost always see a dedicated git server in use.

For open source projects we usually see the remote repository hosted on a public website such as <https://github.com>. Closed source (such as company projects) are very self-hosted with something like <https://gitlab.com>. We will now use utwente's gitlab server to host our source code because you already have an account there and you can share with other students.

Goto <https://gitlab.utwente.nl> and sign in with your student number (e.g. s2037335) and password.

Create repository on Gitlab

The screenshot shows the GitLab user interface. At the top, there is a dark blue navigation bar with the GitLab logo, a search bar, and various utility icons. Below this is a sidebar on the left with a 'Your work' section containing links to Projects, Groups, Issues, Merge requests, To-Do List, Milestones, Snippets, Activity, Environments Dashboard, Operations Dashboard, and Security. The main content area is titled 'Welcome to GitLab' with the tagline 'Faster releases. Better code. Less pain.' Below the title are four cards: 'Create a project' (highlighted with a red border), 'Create a group', 'Explore public projects', and 'Learn more about GitLab'. The 'Create a project' card contains the text: 'Projects are where you store your code, access issues, wiki and other features of GitLab.'

Create repository on Gitlab

The screenshot shows the GitLab web interface for creating a new project. The top navigation bar includes a search bar and utility icons. A left sidebar lists navigation options like 'Projects', 'Groups', and 'Issues'. The main content area is titled 'Create new project' and features four options:

- Create blank project**: Create a blank project to store your files, plan your work, and collaborate on code, among other things. This option is highlighted with a red border.
- Create from template**: Create a project prepopulated with the necessary files to get you started quickly.
- Import project**: Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.
- Run CI/CD for external repository**: Connect your external repository to GitLab CI/CD.

At the bottom, a note states: "You can also create a project from the command line. [Show command](#)"

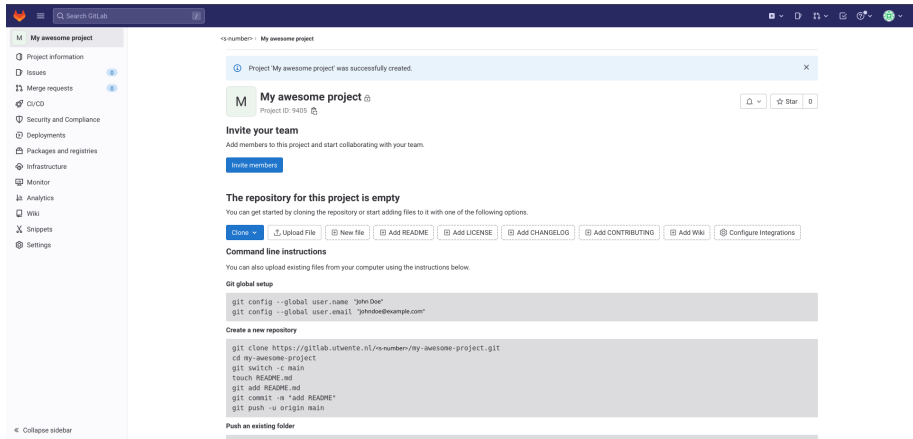
Create repository on Gitlab

The screenshot shows the GitLab interface for creating a new project. The left sidebar contains navigation options like 'Your work', 'Projects', 'Groups', 'Issues', 'Merge requests', 'To-Do List', 'Milestones', 'Snippets', 'Activity', 'Environments Dashboard', 'Operations Dashboard', and 'Security'. The main content area is titled 'Create blank project' and includes a sub-header 'Create blank project' and a description: 'Create a blank project to store your files, plan your work, and collaborate on code, among other things.'

Three numbered callouts highlight specific form fields:

- 1** Points to the 'Project name' input field, which contains the text 'My awesome project'. Below it is a note: 'Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.'
- 2** Points to the 'Visibility Level' section, which has three radio button options: 'Private' (selected), 'Internal', and 'Public'. Each option has a brief description of its access level.
- 3** Points to the 'Project Configuration' section, which includes a checkbox for 'Initialize repository with a README' and a checkbox for 'Enable Static Application Security Testing (SAST)'. Below these are 'Create project' and 'Cancel' buttons.

Create repository on Gitlab



The screenshot shows the GitLab web interface for a newly created repository. The top navigation bar is dark blue with the GitLab logo and search bar on the left, and utility icons on the right. The left sidebar contains a navigation menu for the project, including Project information, Issues, Merge requests, CI/CD, Security and Compliance, Deployments, Packages and registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings. The main content area displays a success message: "Project 'My awesome project' was successfully created." Below this is the project header "My awesome project" with a lock icon, a star icon, and a notification icon. The "Invite your team" section includes the text "Add members to this project and start collaborating with your team." and an "Invite members" button. The "The repository for this project is empty" section provides options to start the repository: Clone, Upload File, New file, Add README, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Wiki, and Configure Integrations. The "Command line instructions" section includes a "Git global setup" block with the following commands:

```
git config --global user.name "John Doe"
git config --global user.email "john@example.com"
```

 and a "Create a new repository" block with the following commands:

```
git clone https://gitlab.utwente.nl/<number>/my-awesome-project.git
cd my-awesome-project
git switch -c main
touch README.md
git add README.md
git commit -m "add README"
git push -u origin main
```

 The "Push an existing folder" section is partially visible at the bottom.

Set up remote tracking locally

On our local repository we will add reference to the remote repository we just created:

```
$ git remote add origin https://gitlab.utwente.nl/<s-number>/<project-name>.git
```

Push to remote

If we look at the Gitlab page we will not see anything we did that. That is because we have not yet pushed our commits to the remote. Let's synchronise these repositories:

```
$ git push -u origin main
```

Note: you may need to provide login information. On a public repository everyone can download but only members can push changes to it.

Look at repository on Github/Gitlab

You will now see your files and README on the remote repository!

The screenshot shows the GitLab interface for a repository named 'Git-Example'. At the top, there is a navigation bar with a search bar and buttons for 'Settings' and 'More information'. Below this, a blue banner indicates that the Auto DevOps pipeline has been enabled. The main content area features a header with the repository name 'Git-Example', a dropdown menu for branches (currently showing 'main'), and buttons for 'History', 'Find file', 'Edit', and 'Code'. A recent commit is displayed, titled 'Merge branch 'separate-files'', authored by '7kasper' 9 hours ago. Below the commit, a table lists files: 'README.md' and 'TUTORIAL.md', both with their last commit information. The 'README.md' file is expanded, showing its content: 'Git Course Summary' and a paragraph about the README file. On the right side, the 'Project information' section shows 5 commits, 1 branch, 0 tags, and 5 KiB of project storage. Below this, the 'README' section is visible, listing links for 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Add Kubernetes cluster', 'Add Wiki', and 'Configure Integrations'. The bottom of the page shows 'Created on'.

Project

- Git-Example
- Pinned
- Issues
- Merge requests
- Manage
- Plan
- Code
- Build
- Secure
- Deploy
- Operate
- Monitor
- Analyze
- Settings
- Help

s2037335 / Git-Example

The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.

Settings More information

Git-Example

main git-example /

History Find file Edit Code

Merge branch 'separate-files'
7kasper authored 9 hours ago

Name	Last commit	Last update
README.md	Merge branch 'separate-files'	9 hours ago
TUTORIAL.md	Move "How to commit" to TUTORIAL....	9 hours ago

README.md

Git Course Summary

This README .md file will contain some markdown text detailing what I learned during the Git Course.

TUTORIAL.md - Basic tutorial on how to commit.

Project information

- 5 Commits
- 1 Branch
- 0 Tags
- 5 KiB Project Storage

README

- Auto DevOps enabled
- Add LICENSE
- Add CHANGELOG
- Add CONTRIBUTING
- Add Kubernetes cluster
- Add Wiki
- Configure Integrations

Created on

Inviting other developers

As previously said; since the repository is public everyone on utwente can see and copy the code. To allow people on private repositories or to allow them to also push changes to the remote you can invite them as members:

The screenshot shows the GitHub interface for a repository named 'Git-Example'. The 'Project members' section is active, and an 'Invite members' dialog box is open. The dialog box contains the following elements:

- 1**: The 'Members' link in the left sidebar navigation menu.
- 2**: The 'Members' link in the top navigation bar.
- 3**: The 'Invite members' button in the top right corner of the 'Project members' section.
- 4**: The 'Username, name or email address' input field in the dialog box, containing the text 'Verzijden, J.C. (Johan, Student B-EE)'.
- 5**: The 'Select a role' dropdown menu in the dialog box, currently set to 'Developer'.
- 6**: The 'Invite' button at the bottom right of the dialog box.

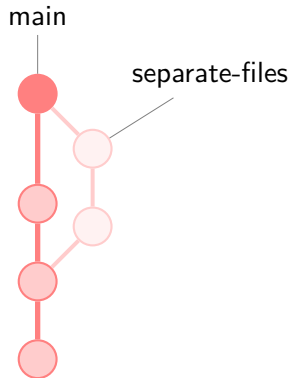
Cloning

The remote is where people base their own local repositories on. If we are invited to someone's repository we can clone this repository onto our local machine. We can then edit code, do commits and eventually push back to the remote.

Clone someone else's repo (after being added as member) or simulate working together by cloning your own repository in a different folder:

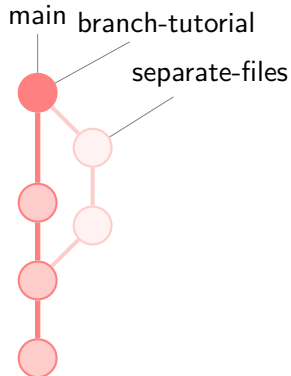
```
$ cd ../  
$ mkdir clone  
$ cd clone  
$ git clone https://gitlab.utwente.nl/<s-number>/<project-  
name>  
$ cd <project-name>
```

Comitting on the clone



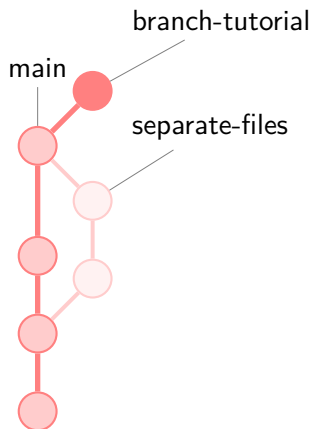
- We will do a commit on the new branch *branch-tutorial* on the clone:
`$ git switch -c branch-tutorial`

Comitting on the clone



- We will do a commit on the new branch *branch-tutorial* on the clone:
`$ git switch -c branch-tutorial`
- Now add a summary how to make and switch to a branch to the *TUTORIAL.md* file.
- Again commit: `$ git commit -am 'Add branching tutorial'`

Comitting on the clone



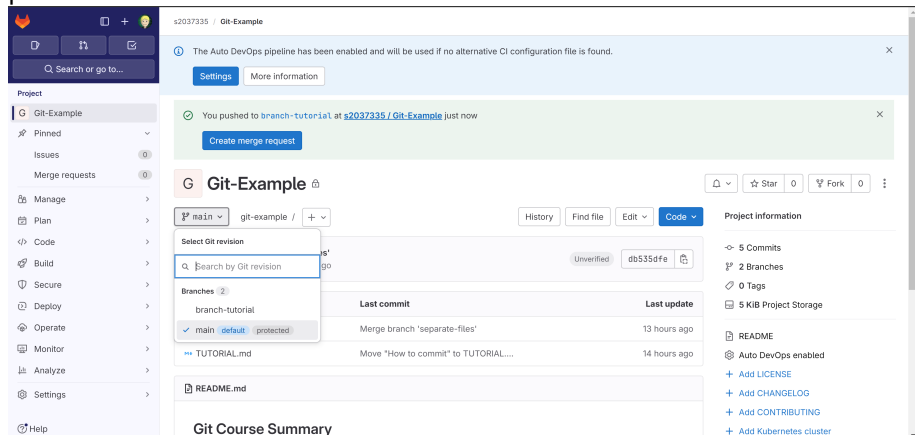
- We will do a commit on the new branch *branch-tutorial* on the clone:
`$ git switch -c branch-tutorial`
- Now add a summary how to make and switch to a branch to the *TUTORIAL.md* file.
- Again commit: `$ git commit -am 'Add branching tutorial'`
- We now see a new commit on *branch-tutorial* in the clone repository.
- *Note: Although the separate-files history is embedded in the main after the merge commit the branch itself is not known as it was never pushed to or pulled from the remote.*

Pushing from the clone

We commit our new branch from the clone to the remote repository:

```
$ git push origin branch-tutorial
```

After this we can see that on the website we can switch between the pushed branches:



The screenshot shows the GitHub interface for a repository named 'Git-Example'. The 'branch-tutorial' branch is selected in the 'Select Git revision' dropdown menu. The main content area displays the commit history for this branch, showing a merge of 'branch 'separate-files'' and a move of 'TUTORIAL.md'. The 'Project information' sidebar on the right shows 5 commits, 2 branches, and 0 tags. The 'README.md' file is also visible in the file list.

Project

- Git-Example
- Pinned
- Issues
- Merge requests
- Manage
- Plan
- Code
- Build
- Secure
- Deploy
- Operate
- Monitor
- Analyze
- Settings
- Help

2037335 / Git-Example

The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.

You pushed to `branch-tutorial` at `s2037335 / Git-Example` just now

Create merge request

Git-Example

main git-example / +

Select Git revision

Search by Git revision

Unverified db535dfe

Branches	Last commit	Last update
branch-tutorial	Merge branch 'separate-files'	13 hours ago
main	protected	
TUTORIAL.md	Move "How to commit" to TUTORIAL....	14 hours ago

Project information

- 5 Commits
- 2 Branches
- 0 Tags
- 5 KIB Project Storage
- README
- Auto DevOps enabled
- Add LICENSE
- Add CHANGELOG
- Add CONTRIBUTING
- Add Kubernetes cluster

README.md

Git Course Summary

Merge request

Instead of merging within the commandline we can create a merge request (often also called pull request). This is very useful for working together. Before actually bringing the *branch-tutorial* branch changes into the *main* version of our summary project we first create the request for this. This allows us to write some text explaining, discussing this with comments and also have other people review the changes you suggest.

Note: On open source projects you are often not a member so you cannot push to these repositories straight away. You can however create a so-called fork of the project. This is your own copy where you have all the access rights. After pushing to your fork you can create a merge-request to push the changes 'upstream' to the actual repository.

Let's try it!

Open merge request

The screenshot shows a GitLab interface for a merge request. The page title is "My awesome project fork" with a project ID of 9407. It shows 5 commits, 1 branch, 0 tags, and 10 KB of project storage. A recent commit by Johan Verzijden is highlighted, titled "Change the text colour to something amazing". The merge request is currently open, and a red box highlights the "Create merge request" button. The table below lists the files involved in the merge request.

Name	Last commit	Last update
index.html	Merge branch 'add-learning-git-project'	1 week ago
style.css	Change the text colour to something amazing	1 minute ago

Open merge request

My awesome project fork

My awesome project fork > Merge requests > New

New merge request

From `/my-awesome-project-fork:main` into `/my-awesome-project:main` [Change branches](#)

Title (required)

Mark as draft
Drafts cannot be merged until marked ready.

Description

[Write](#) [Preview](#)

B I H L U P

Describe the goal of the changes and what reviewers should be aware of.

Supports [Markdown](#). For [quick actions](#), type `/`.

[Add description templates](#) to help your contributors to communicate effectively!

Assignees

Unassigned [Assign to me](#)

Reviewers

Unassigned

Approvals are optional.

[Approval rules](#)

Milestone

Select milestone

Labels

« Collapse sidebar

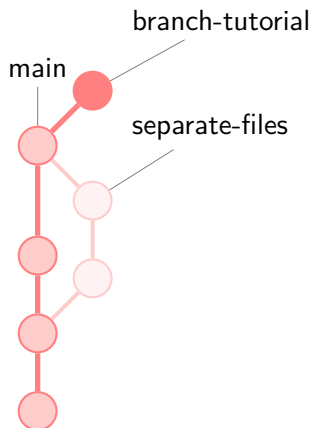
Open merge request

The screenshot shows the GitLab web interface for a project named "My awesome project fork". The left sidebar contains navigation options: Project information, Repository, Issues (1), Merge requests (2), CI/CD, Security and Compliance, Deployments, Packages and registries, Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings. The main content area is titled "Create merge request" and includes sections for "Approvals are optional", "Milestone" (with a "Select milestone" dropdown), "Labels" (with a "Labels" dropdown), "Merge request dependencies" (with a text input for URLs and a note that references should be in the form of path/to/project/merge_request_id), "Merge options" (with a checkbox for "Squash commits when merge request is accepted"), and "Contribution" (with a checkbox for "Allow commits from members who can merge to the target branch"). At the bottom of the form, there are two buttons: "Create merge request" (highlighted with a red box) and "Cancel". Below the form, there is a "Commits" section showing a single commit from 20 Jun, 2023, titled "Change the text colour to something amazing" by Johan Verzijden, with a commit hash of 40fccf80.

Review and accept merge request

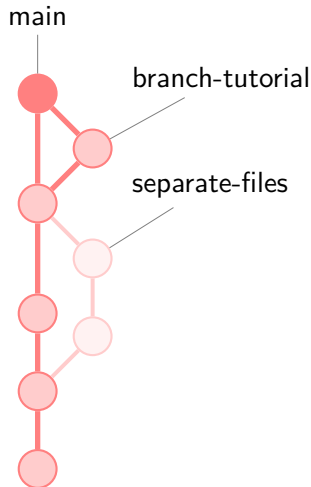
The screenshot displays the GitLab Merge Request interface for a project named "My awesome project". The main heading is "Change the text colour to something amazing". The merge request is requested to merge into the "main" branch, which was 6 days ago. The "Changes" tab is selected and highlighted with a red box. Below the tab, there are buttons for thumbs up (0), thumbs down (0), and a refresh icon. A dropdown menu shows "Approval is optional". A green "Ready to merge!" status is displayed. Below this, there are options for "Squash commits" and "Edit commit message". A note states: "The source branch is 2 commits behind the target branch. 1 commit and 1 merge commit will be added to main." A blue "Merge" button is highlighted with a red box. The "Activity" section is visible below, with a "Write" tab selected and a rich text editor containing the text "Write a comment or drag your files here...". The right sidebar shows various settings like "Mark as done", "Assignees", "Reviewers", "Labels", "Milestone", "Time tracking", "Lock merge request", "Notifications", and "1 Participant".

Performing the merge



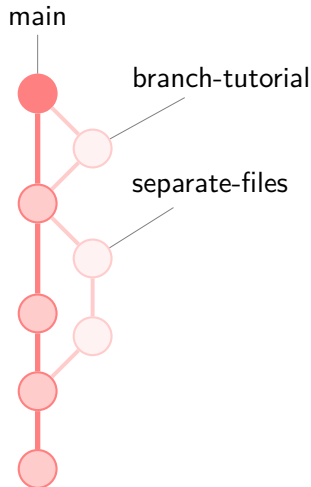
- On the website create a merge request. Merge *branch-tutorial* into *main*.
- Review and merge the request.

Performing the merge



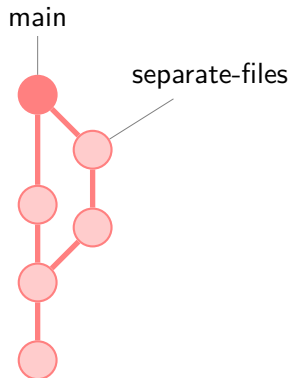
- On the website create a merge request. Merge *branch-tutorial* into *main*.
- Review and merge the request.
- Now again a merge commit is created on the *main* branch.
- If we know we don't need the *branch-tutorial* branch anymore we can 'Delete source branch' from the remote.

Performing the merge



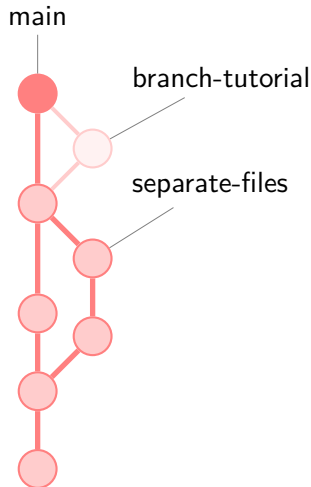
- On the website create a merge request. Merge *branch-tutorial* into *main*.
- Review and merge the request.
- Now again a merge commit is created on the *main* branch.
- If we know we don't need the *branch-tutorial* branch anymore we can 'Delete source branch' from the remote.
- We see the remote no longer holds the source branch but all changes and history are embedded from the merge.

Pulling from the origin



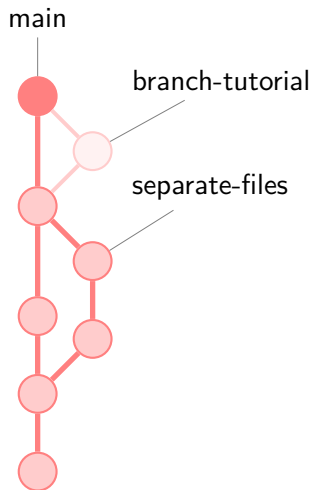
- Let's switch back to our first local repository:
`$ cd ../../<project-name>`
- Now of course we don't know about the changes of the remote yet. Let's reel them in:
`$ git pull origin main`

Pulling from the origin



- Let's switch back to our first local repository:
`$ cd ../../<project-name>`
- Now of course we don't know about the changes of the remote yet. Let's reel them in:
`$ git pull origin main`
- Yes! Now our changes are also in the other local branch!
- *Note: In this repository we still of course have the separate-files branch as we never deleted it locally. We do not have the branch-tutorial branch as we have never pulled it from the remote.*

Reverting a commit

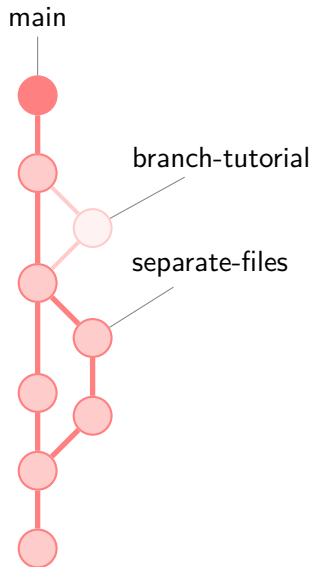


- Imagine that the last change screwed up our program and we want to delete it. Git helps a lot by recording all changes so we can just revert the change.
- First find out the hash of the commit that introduced the mistake. You can do this on the website or perhaps with:

```
$ git log
```
- In my case it seemed the 'Add branching tutorial' was wrong. Its hash starts with 'd7f775'. We can revert it using:

```
$ git revert d7f775
```

Reverting a commit



- Imagine that the last change screwed up our program and we want to delete it. Git helps a lot by recording all changes so we can just revert the change.
- First find out the hash of the commit that introduced the mistake. You can do this on the website or perhaps with:

```
$ git log
```
- In my case it seemed the 'Add branching tutorial' was wrong. Its hash starts with 'd7f775'. We can revert it using:

```
$ git revert d7f775
```
- This creates a new commit on `main` that does the opposite changes of the commit specified. It essentially negates it.

Never forget

Reverting creates a new commit. History never gets deleted this way and we can even revert the revert. Of course to update to our other developers we have to push to the origin and then pull on the other machines again.

Note: While it is not advised to muck about in the history of git commits you can using git reset for instance. A use case might be that you accidentally committed a password to a shared repository and also want to remove this from history. As these actions are more uncommon we will discuss them only in the intermediate course.

Done!

This concludes the practical for the basic git course. You have now used all important and most essential git commands and concepts, congratulations!

There is a lot more that Git can do. It can help you find bugs by doing a binary search on your code, there are commands to easily help you find changes in history (so you know who to blame). You can work together with multiple upstream branches, secure your code with signed PGP keys, etc. If you want to learn more about this search the web or come to our intermediate course! :-)

Common Git Commands - Cheat sheet

git ...

- `init`
- `add [<filenames>]`
- `commit [-m '<message>']`
- `switch [-c] <branch name>`
- `log [--graph --oneline]`
- `push`
- `pull`
- `fetch`
- `clone <path/URL>`
- `blame <file> [--color-by-age]`
- `revert <commit>`
- `reset [--hard] <commit>`

Sources for future reference

- `git help <command>`
- `man gittutorial`
- gitimmersion.com

Think of a question later on? Feel free to reach out to us!

MasterCLASS

masterclass@scintilla.utwente.nl

Kasper Müller

kasperm@scintilla.utwente.nl

Johan Verzijden

johanv@scintilla.utwente.nl