Installation

- On Windows: Download git here: https://git-scm.com/download/win and install it
- On Linux: execute ‘sudo apt install git’ in the terminal

Configuration

Basics

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

Run these commands without –global to apply changes only for local repository.

Advanced

- Change from default editor by executing `$ git config --global core.editor vim`
- Create aliases by:
  - `git config --global alias.unstage 'reset HEAD --'`: Introduces new command unstage
  - `git config --global alias.hist 'log --pretty=format:"%h - %an, %ar : %s" --graph'`: Is an pretty alternative to git log.
- Check the configuration by `$ git config --list`

Getting started

Create an directory

Select an directory of your choice e.g. ~/Desktop/git-training/, open the terminal there and type

```
$ git init
```

Congratulations you have your first git repository!

Committing files

Create some text files e.g. `hallo.txt` or `hello_world.py`. To start version-controlling them add the file to git by using the command

```
$ git add filename
```

You can also add multiple files. E.g. you want to add all text files the do

```
$ git add *.txt
```

Now we can commit these changes by typing

```
$ git commit
```

This will open your preferred text editor where you can type in a commit message.

Commit messages

Commit messages are important

Why are the so important? Because it is much easier to track bugs with it, they describe the process of development, make it easier for others to work with your code (e.g. code review) and will help you in two years to understand what you did. They to some extend replace your labbook in its function.

How to write commit messages

- Use present tense and use imperative
- Tell why you did changes.
- Limit yourself to 50 characters in the first line.
- Wrap the body at 72 characters

Good commit messages

For short commits that need only few explanation it is convenient to use

```
$ git commit -m "Fix typo in introduction to user guide"
```

For long commit message you use

```
$ git commit
```

which opens a text editor and lets you describe your changes in detail. Do not forget the line between header and body of the commit message.

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too
 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here
```

Hall of shame

Here you see some examples of bad commit messages.

- "bug fix", "more work", "minor changes": Bad because it contains no useful information
- "Change X constant to be 10": Bad as it does not tell why. What was changed is easy to find out by git diff.
- "super long commit message goes here, something like 100 words and lots of characters woohoo!": Bad because it is unreadable.

Storing changes on remote server

To store changes on a remote server e.g. Github you need to push your repository there. If you are doing this for the first time you first need to define where to push your repository to. Use

```
$ git remote add origin https://github.com/GerJuli/introduction-to-git.git
```

The URL can be replaced with any URL that points to an existing repository. This can even be a USB drive! Now push by using `git push -u origin master`. With this command you created an upstream to your remote 'origin' where you branch 'master' will be pushed. As this is set up you can use `git push` from now on.

Cloning existing repositories

Downloading existing repositories is called 'cloning'. You can do this by using

```
$ git clone https://github.com/GerJuli/introduction-to-git.git
```

That is it. Now you can start working on it.

Workflow

Check status

With

```
$ git status
```

you can check the status of your current directory. This will look like this:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Add file

If you now add a file to the repository e.g. README then the status changes.

```
$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

      README

      nothing added to commit but untracked files present (use "git add" to track)
```

You can also run

```
$ git diff
```

which will show you the changes that where made.

Now track the file by adding it to the staging area via

```
$ git add README
```

If you want to commit only a part of the changes you made use

```
git add -p
```

git will ask you for every "hunk"(part of the file) if you want to add it to the staging area. The main options are:

- y stage this hunk for the next commit
- n do not stage this hunk for the next commit
- q quit; do not stage this hunk or any of the remaining hunks

Committing

The changes are now ready to be committed

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```
      new file:   README
```

Now commit this change with a suitable commit message. You forgot what you changed and `git diff` does not work? If you already added the file to the staging area you can run

```
$ git diff --staged
```

Push to remote

Push this now to your repository with

```
$ git push
```

Show last changes

```
$ git log
```

shows you a log of recent commit messages.

If you want to have a look at the changes source use

```
$ git log -p
```

A much prettier version is

```
$ git log --pretty=format:"%h - %an, %ar : %s" --graph
```

Get latest changes

An amazing thing of git is to work together with others. This guide will not go into the details of this, yet you will often have to get the latest changes of a project.

You can do this easily by using

```
$ git pull
```

which is a short version of two different commands. It first executes `git fetch` that downloads the latest changes. Then it runs `git merge FETCH_HEAD` which tries to apply the newest changes to your local repository. This is also the difficulty of this as any file that was changed locally and in the remote repository will cause a *merge conflict* as git does not know which file/parts of the file to keep. The resolution of such a conflict can be complicated and goes beyond the intentions of this guide.

A brief overview can be found here.

Gitignore

Sometimes you do not want git to track every file in your repository. Typically this applies for log files, configuration files (where maybe passwords are stored) and compiled sources. For this purpose you can create the file `.gitignore` in your repository. This file will be read by git. An example `.gitignore` looks like this:

```
# ignore all .log files
*.log

# ignore all files in any directory named config
build/

# but do track sample.log, even though you're ignoring .log files above
!sample.log

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/
```

You can generate gitignore files here.

Undoing things

Make changes to the last commit

You forgot to add one file to your commit? You are unhappy with your commit message? Use `git --amend`!

```
$ git commit -m 'Pretty commit message'
$ git add forgotten_file
$ git commit --amend
```

Do not do this if you already pushed to remote.

Unmodifying a Modified File

You made changes to a file but the changes messed everything up? You can always go back to the version of the last commit by

```
git checkout -- filename
```

Warning: This is dangerous as you delete all changes that you made locally. Do NOT use this command unless you are absolutely sure what you are doing.

Time machine

Something went terribly wrong. When using git this is no problem. Just use `reflog` and select the commit where still everything was alright.

```
$ git reflog
9bff138 HEAD@{0}: commit: Document limitations of this guide
8d9dac8 HEAD@{1}: commit (amend): Adjust script to nice print
[...]
2b66f53 HEAD@{7}: commit: Introduce git log
8835c33 HEAD@{8}: commit: Fix of formating of headings
2f0f500 HEAD@{9}: commit: Improve order of comments on workflow
249dc14 HEAD@{10}: commit (amend): Explain git diff
f5ba1d3 HEAD@{11}: commit: Explain git diff
0a2ce1c HEAD@{12}: commit: Remove trailing whitspaces
735511f HEAD@{13}: commit (initial): Initial commit
git reset HEAD@{index_where_everything_was_fine}
```

Warning: This is dangerous as you delete all changes that you made locally. Do NOT use this command unless you are absolutely sure what you are doing.

USB sticks as remote

Sometimes (e.g. you are working on a machine that is not connected to the internet) it can be helpful to use an USB drive as remote.

How to prepare the drive

Go to the USB drive

```
$ cd /media/username/git-stick/
```

Create a directory for your repository

```
$ mkdir my-repo
```

Initialize the repo

```
$ cd my-repo
$ git init --bare
```

How to push to a USB drive

To push to a USB drive you need to add a remote. Go to your local repository and use

```
$ git remote add usb /media/username/git-stick/my-repo/
```

Be aware that your stick will have a different name and will be at a different location, depending on your operating system. You named the remote “usb” here, of course you can change that name.

Now push your repo

```
$ git push usb
```

To list all repositories just type

```
$ git remote
origin
usb
```

If you want to push to the USB drive by default you can use

```
$ git push -u usb master
Branch 'master' set up to track remote branch 'master' from 'usb'
```

What this guide does not cover

- Branching
  - Creating branches
  - Merging branches (and merge conflicts)
  - Remote branches
  - Tagging
  - Forks
- Usage of Github
- Hooks

Further reading

- Official git website Git SCM: Extensiv documentation
- Oh shit Git: Help when you messed up.