



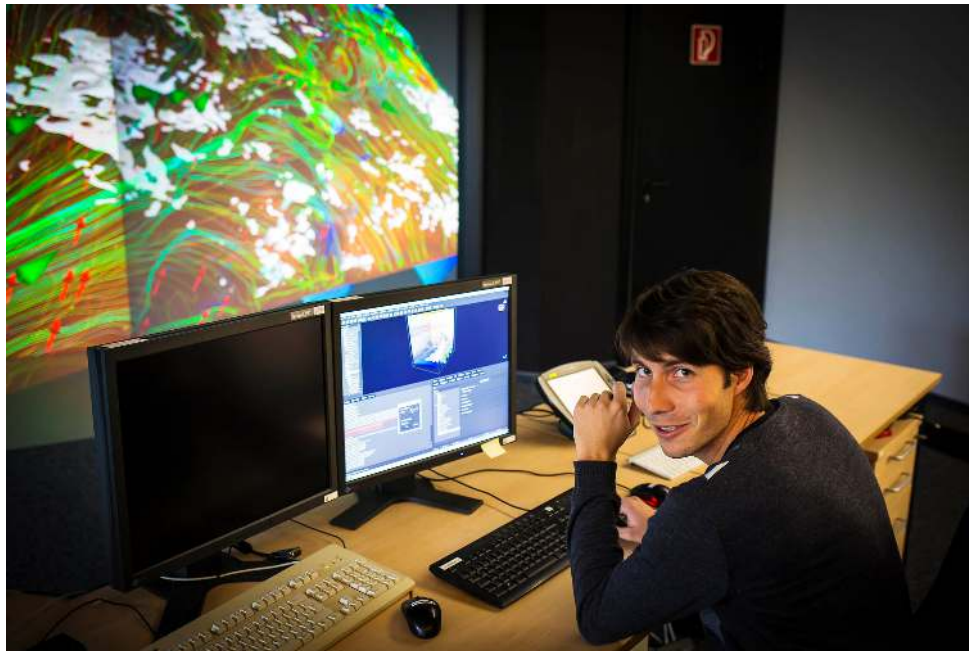
## Hydroinformatik II – V1/Ü1 Software-Grundlagen - Git, Qt

Lars Bilke, Environmental Informatics

Dresden, 05.04.2019

# About me

- Lars Bilke
- Computer Science
- Since 2008 at UFZ / ENVINF
- Visualization Center TESSIN VISLab
- Software engineer OpenGeoSys



# Overview

- Version control system
  - Git by example
  - Exercise: Online tutorial
  - Hosting services
  - Exercise: Get example files from GitHub
- Exercise / HW: Qt Creator installation



git-scm.org

# VERSION CONTROL SYSTEMS

# Version control systems: What

- „Database“ which takes snapshots (version) of files
- Tracks time, changes, authors
- Some content and images from [www.git-tower.com/learn](http://www.git-tower.com/learn)



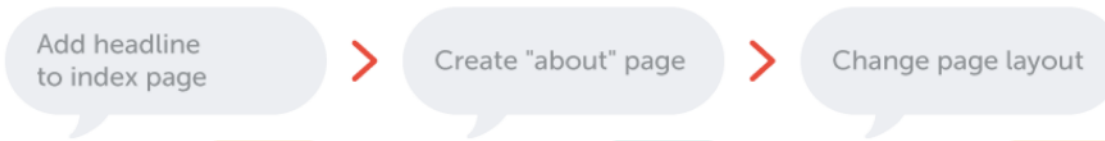
Time



Your project



VCS



index.html **modified**  
<h1>Headline</h1>  
...

about.html **created**  
<html>  
<head>  
...  
photo.png **created**

about.html **modified**  
<div>new content</div>  
...



**HELMHOLTZ**  
CENTRE FOR  
ENVIRONMENTAL  
RESEARCH – UFZ

# Version control systems: What

- Independent of project / technology / tools
- Best suited for all kinds of plain text files
  - LaTeX, Markdown, TXT, CSV, Python scripts, source code, ...
- Can handle binary files too, e.g. images

# Version control systems: Why

- Collaboration
- Storing versions
- Restoring previous versions / undo
- Understanding history of a project
- Backup

# Version control systems: Collaboration

- „Classic“ collaboration via shared folders or emailing files back and forth is error prone
  - Manually (shouting through the office) „lock“ files
  - Merging changes from several emails
  - Where is the latest version?
  - You will lose / overwrite changes!
- With VCS anybody can work on any file at any time
  - Merge changes into a common version
  - Latest version is in a common central place



# Version control systems: Versions

- „Classic“ approach:
  - How do you handle versions? By file name postfix?
  - What has changed between *paper\_v56.doc* and *paper\_v61.doc*?
  - What if you want to have variants of a document?
- VCS:
  - Just gives you one state of a project
  - Handles versions for you
  - You can go to any previous version
  - Parallel variants are handled too („branches“)

# Version control systems: Restoring

- Restoring a file or whole project
  - You can't mess something up!
- Undo changes

# Version control systems: History

- VCS requires a short description on changes
  - High-level overview
- Detailed changes for the entire history

# Version control systems: Backup

- VCS server component acts as a backup
- Collaborators act as backups too

# Version control systems: Definitions

## ❖ Repository

- „Database“ stores all files, its versions and metadata

## ❖ Working directory

- Set of files of your project on your PC

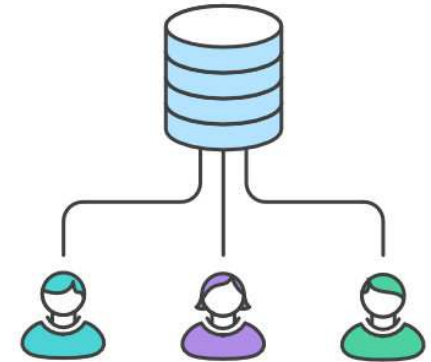
## ❖ Commit

- Set of changes to files

# Version control systems

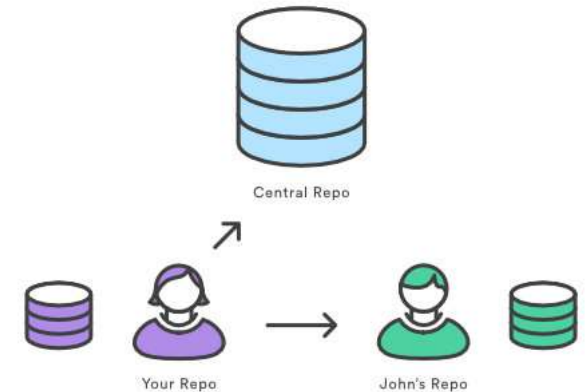
## ■ Centralized

- Subversion – [subversion.apache.org](http://subversion.apache.org)
  - UFZ provides server
- Perforce (\$) – [www.perforce.com](http://www.perforce.com)



## ■ Distributed

- Git – [git-scm.com](http://git-scm.com)
- Mercurial – [www.mercurial-scm.org](http://www.mercurial-scm.org)



Images from [www.atlassian.com/git](http://www.atlassian.com/git)



Version control system


**GIT**

# Version control systems: Git

- Distributed is better than centralized
  - No server necessarily needed / works offline / faster
  - Multiple backups
- Widely used / lots of tools / software
- Many hosting provider (free / \$)
- Powerful / flexible / does not impose a specific workflow
  - But also harder to learn / more concepts



# Git: Getting started

- Git is a command line tool, no GUI!
  - Maybe harder to learn but *you get what you type*
- Installer: [git-scm.com/download/win](http://git-scm.com/download/win)
  - Git Bash, Git Gui 

```
MINGW32:~/git
Welcome to Git (version 1.8.3-preview20130601)

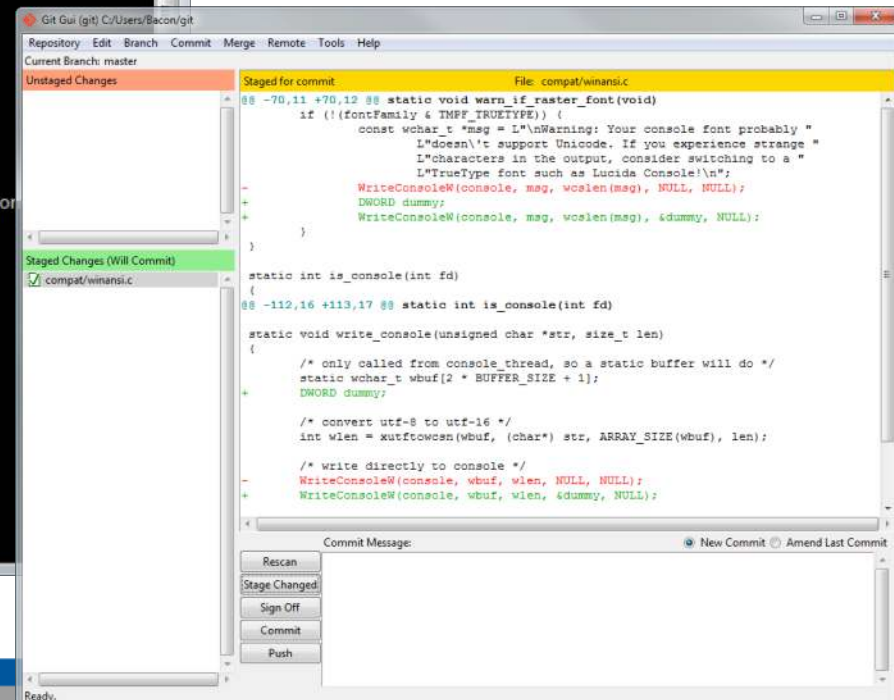
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Bacon@BACON ~
$ git clone https://github.com/msysgit/git.git
Cloning into 'git'...
remote: Counting objects: 177468, done.
remote: Compressing objects: 100% (52057/52057), done.
remote: Total 177468 (delta 133396), reused 166093 (delta 123576)
Receiving objects: 100% (177468/177468), 42.16 MiB | 1.84 MiB/s, done
Resolving deltas: 100% (133396/133396), done.
Checking out files: 100% (2576/2576), done.

Bacon@BACON ~
$ cd git

Bacon@BACON ~/git (master)
$ git status
# On branch master
nothing to commit, working directory clean

Bacon@BACON ~/git (master)
$
```



```
Repository: Edit Branch Commit Merge Remote Tools Help
Current Branch: master

Unstaged Changes
Staged for commit: File: compat/winansic.c
@@ -70,11 +70,12 @@ static void warn_if_raster_font(void)
    if (!(fontFamily & IMPF_TRUETYPE)) {
        const wchar_t *msg = L"\nWarning: Your console font probably "
            L"doesn't support Unicode. If you experience strange "
            L"characters in the output, consider switching to a "
            L"TrueType font such as Lucida Console!\n";
-       WriteConsoleW(console, msg, wcslen(msg), NULL, NULL);
+       DWORD dummy;
+       WriteConsoleW(console, msg, wcslen(msg), &dummy, NULL);
    }
}

Staged Changes (Will Commit)
[x] compat/winansic.c
static int is_console(int fd)
{
@@ -112,16 +113,17 @@ static int is_console(int fd)
static void write_console(unsigned char *str, size_t len)
{
    /* only called from console_thread, so a static buffer will do */
    static wchar_t wbuf[2 * BUFFER_SIZE + 1];
+   DWORD dummy;

    /* convert utf-8 to utf-16 */
    int wlen = wutf8towcs(wbuf, (char*) str, ARRAY_SIZE(wbuf), len);

    /* write directly to console */
-   WriteConsoleW(console, wbuf, wlen, NULL, NULL);
+   WriteConsoleW(console, wbuf, wlen, &dummy, NULL);
}

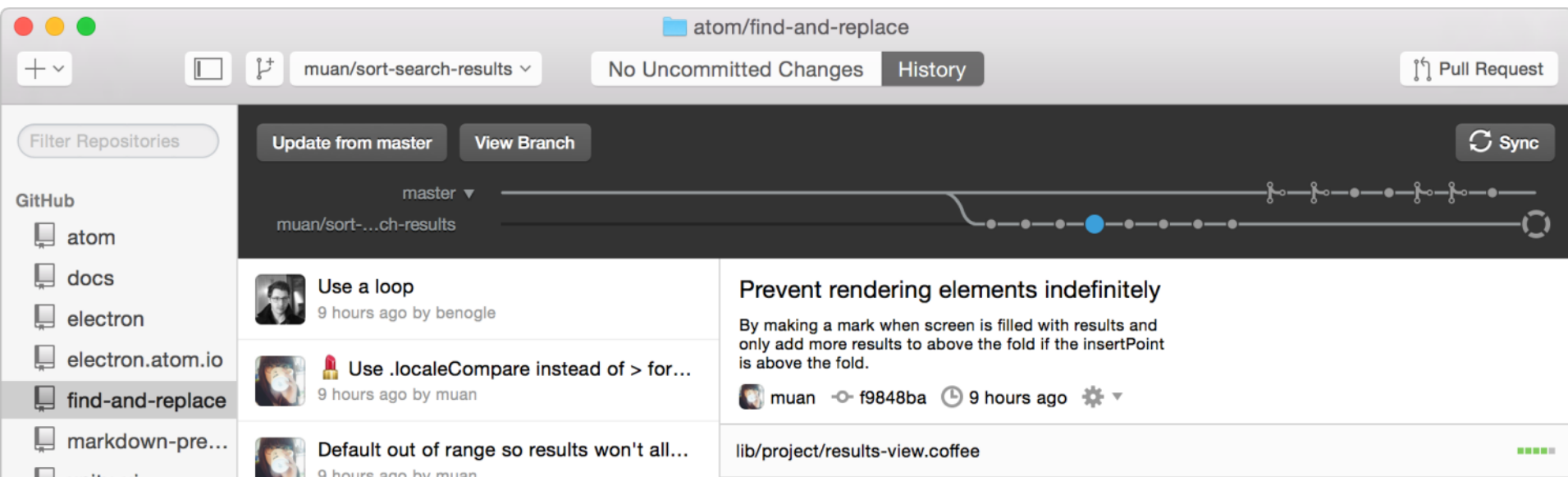
Commit Message: [New Commit] [Amend Last Commit]

Rescan
Stage Changed
Sign Off
Commit
Push

Ready.
```

# Git: Graphical tools

- GitHub Desktop (free) – [desktop.github.com](https://desktop.github.com)
- Win / Mac
- Clean and simple interface



# Git: Graphical tools

- SourceTree (free) – [www.sourcetreeapp.com](http://www.sourcetreeapp.com)
- Win / Mac
- Full featured

The screenshot displays the SourceTree graphical tool interface for a Git repository named 'Sparkle (Git)'. The top toolbar includes icons for View, Commit, Checkout, Reset, Stash, Add, Remove, Add/Remove, Fetch, Pull, Push, Branch, Merge, Tag, Show in Finder, and Git Flow. The main window is divided into several sections:

- FILE STATUS:** Shows 'Working...' with 12 changes.
- BRANCHES:** Lists 'master' and several 'sparkle-1.5b' branches.
- TAGS:** Lists 'sparkle-1.5b1' through 'sparkle-1.5b6'.
- REMITES:** Lists 'SUHost.h' and 'SUHost.m'.
- STASHES:** Lists 'SUHost.m'.
- SUBMODULES:** Empty.
- SUBTREES:** Empty.

The central area shows a commit history table with columns for Graph, Commit, Description, Author, and Date. The current commit is e27828d, described as 'Merge pull request #144 from mattstevens/custom-defaults...'. Below the table, a diff view for 'SUHost.m' is shown, highlighting a hunk of code (lines 25-33) that updates the 'SULog' message and the 'defaultsDomain' logic.

Graph	Commit	Description	Author	Date
	f341d0c	Corrected minor spelling errors	Christian Zachariasen <c...>	Feb 22, 2012, 11:35 AM
	e27828d	Merge pull request #144 from mattstevens/custom-defaults...	Andy Matuschak <andy@...>	Feb 14, 2012, 5:52 PM
	44cf0d0	Merge pull request #145 from mattstevens/install-on-quit-...	Andy Matuschak <andy@...>	Feb 14, 2012, 5:48 PM
	6be5568	Clean up after install on quit updates	Matt Stevens <matt@allo...>	Feb 13, 2012, 3:02 AM
	29b7321	Support a custom user defaults domain	Matt Stevens <matt@allo...>	Feb 11, 2012, 10:31 PM
	449b00e	Merge pull request #139 from mattstevens/pathToRelaunch	Andy Matuschak <andy@...>	Feb 9, 2012, 6:37 AM
	4c25737	Merge pull request #142 from mzch/master	Andy Matuschak <andy@...>	Feb 9, 2012, 6:14 AM
	88d4b34	Improved Japanese translations	Koichi MATSUMOTO <mz...>	Feb 8, 2012, 6:37 AM

```
25 25 SULog(@"Sparkle Error: the bundle being updated at %@
26 +
27 + defaultsDomain = [[bundle objectForKey:@"SUDef
28 + if (!defaultsDomain)
29 + defaultsDomain = [[bundle bundleIdentifier] retain];
```

# Git: Getting started (git init)

- Tell git who you are
  - Start Git Bash
  - Type `git config --global user.name <name>`
  - `git config --global user.email <email>`
- Create a repository (start a new project)
  - Create a directory, `cd` to it, `git init`
  - Repo is hidden inside `.git`-folder
- Create a file, write something ...
- ❖ File status
  - Untracked: not under version control, git does not watch
  - Tracked: under version control, git watches for changes



# Git: Getting started (git add)

## ■ Commit changes

- Tell git which changed files should be included:
  - `git add file.txt`
- Does two things:
  - Sets file.txt to be *tracked*
  - Adds file.txt to the *staging area*

## ❖ Tracked

- Git is managing (versioning) this file

## ❖ Staging area

- Kind of a buffer between working directory and the repository

# Git: Getting started

## ❖ Staging area

### Working Copy

Your Project's Files



Git watches tracked files for new local modifications...

### Tracked (and modified)



If a file was modified since it was last committed, you can stage & commit these changes



Changes that are **not staged** will not be committed & remain as local changes until you stage & commit or discard them

### Untracked



If a file was modified since it was last committed, you can stage & commit these changes

### Staging Area

Your Project's Files



Changes that were added to the Staging Area will be included in the next commit

### Local Repository

Your Project's Files



All changes contained in a commit are saved in the local repository as a new revision

# Git: Getting started (git commit)

- With `git commit` all files / changes in the staging area make up a version
- A commit should be described shortly
  - `git commit -m „Added a simple text file.“`

## ❖ Commit

- Set of changes
- Author
- Timestamp
- Hash (unique identifier)
- Parent commit

```
commit 215f52ccd4b7f1a449e15f3996a0f2fcdccb16cd
Author: Lars Bilke <lars.bilke@ufz.de>
Date: Thu Oct 22 12:46:53 2015 +0200

    Added a simple text file.

diff --git a/file.txt b/file.txt
new file mode 100644
index 0000000..b72e979
--- /dev/null
+++ b/file.txt
@@ -0,0 +1 @@
+A simple text file.
```

# Git: Getting started (git diff)

- Now make more changes, review & commit them
  - Edit file
  - Show changes with `git diff`

```
@@ -1 +1,2 @@  
-A simple text file.  
+A simple text file to demonstrate  
+how to get started with git.
```



# Git: Getting started (git status)

## ■ Check current status

- `git status`

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       modified:   file.txt

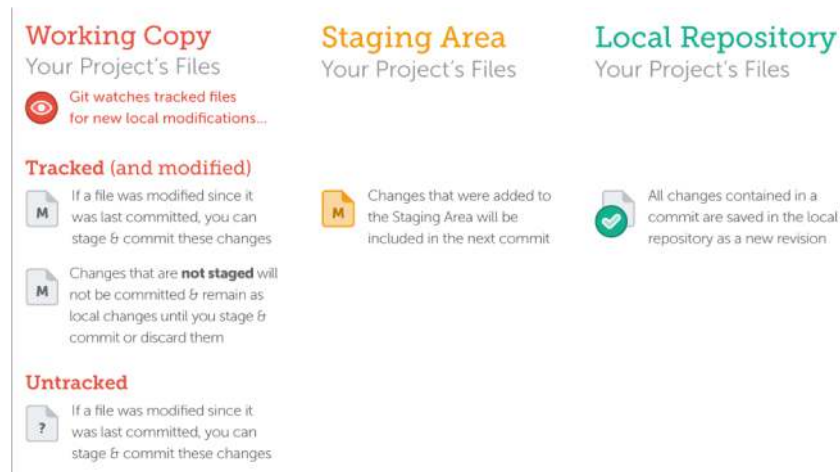
no changes added to commit (use "git add" and/or "git commit -a")
```

- `git add file.txt`
- `git status`

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

       modified:   file.txt
```

- `git commit -m "..."`



# Git: Getting started (git log)

- Show history with `git log`

```
commit bcf61872cf9063297cdf389f06e4e58b36237e8a
Author: Lars Bilke <lars.bilke@ufz.de>
Date: Thu Oct 22 12:57:50 2015 +0200

    More precise file description.

commit 215f52ccd4b7f1a449e15f3996a0f2fcdccb16cd
Author: Lars Bilke <lars.bilke@ufz.de>
Date: Thu Oct 22 12:46:53 2015 +0200

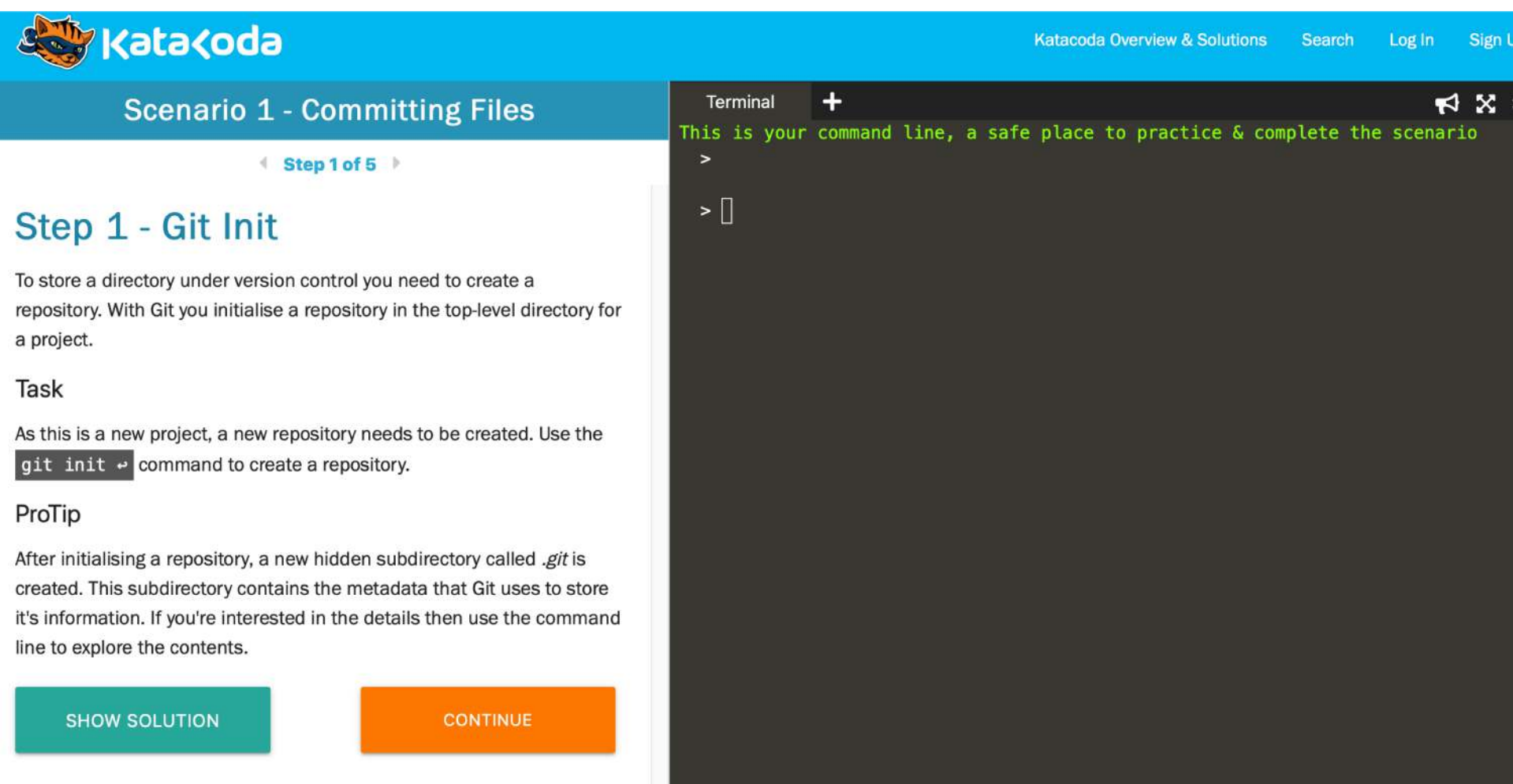
    Added a simple text file.
```

- `git log --oneline`

```
bcf6187 More precise file description.
215f52c Added a simple text file.
```

# Git: Getting started (exercise)

- <https://www.katacoda.com/courses/git>
- Do scenarios 1 and 2



The screenshot shows the Katacoda interface for a Git exercise. The top navigation bar is blue with the Katacoda logo on the left and links for "Katacoda Overview & Solutions", "Search", "Log In", and "Sign U" on the right. Below the navigation bar, the page title is "Scenario 1 - Committing Files" with a sub-header "Step 1 of 5". The main content area is divided into two columns. The left column contains the exercise instructions, and the right column contains a terminal window.

**Scenario 1 - Committing Files**

◀ Step 1 of 5 ▶

## Step 1 - Git Init

To store a directory under version control you need to create a repository. With Git you initialise a repository in the top-level directory for a project.

### Task

As this is a new project, a new repository needs to be created. Use the `git init` command to create a repository.

### ProTip

After initialising a repository, a new hidden subdirectory called `.git` is created. This subdirectory contains the metadata that Git uses to store its information. If you're interested in the details then use the command line to explore the contents.

[SHOW SOLUTION](#) [CONTINUE](#)

**Terminal** +

This is your command line, a safe place to practice & complete the scenario

```
>  
> []
```

# Git: Syncing

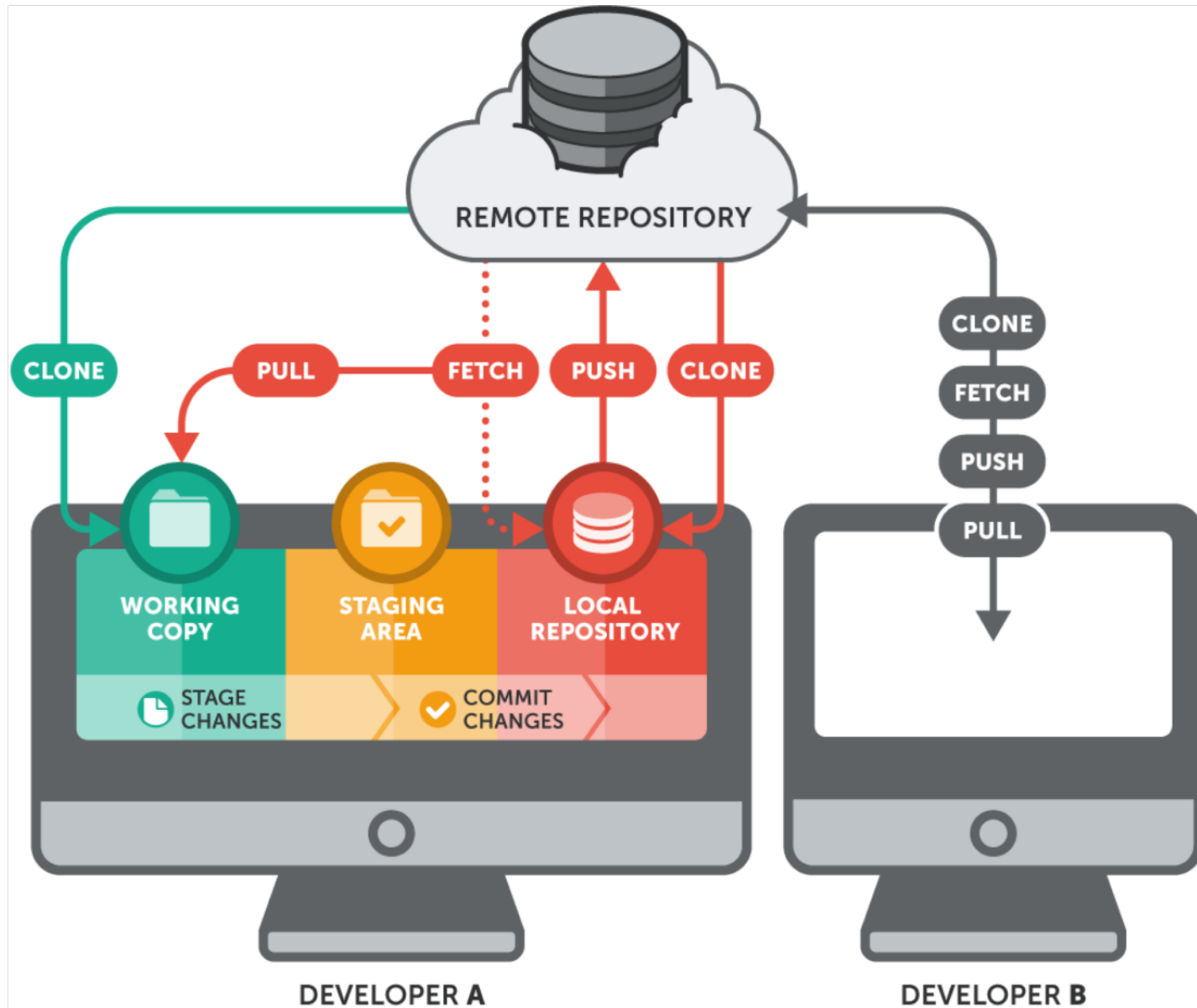
## ❖ Local repository

- Hidden `.git`-folder in working directory
- The one you interacted with

## ❖ Remote repository

- Typically on a server on the internet
- Has no working directory, just contents of `.git`
- People use remote repos to share and exchange data

# Git: Syncing



# Git: Syncing (git remote add)

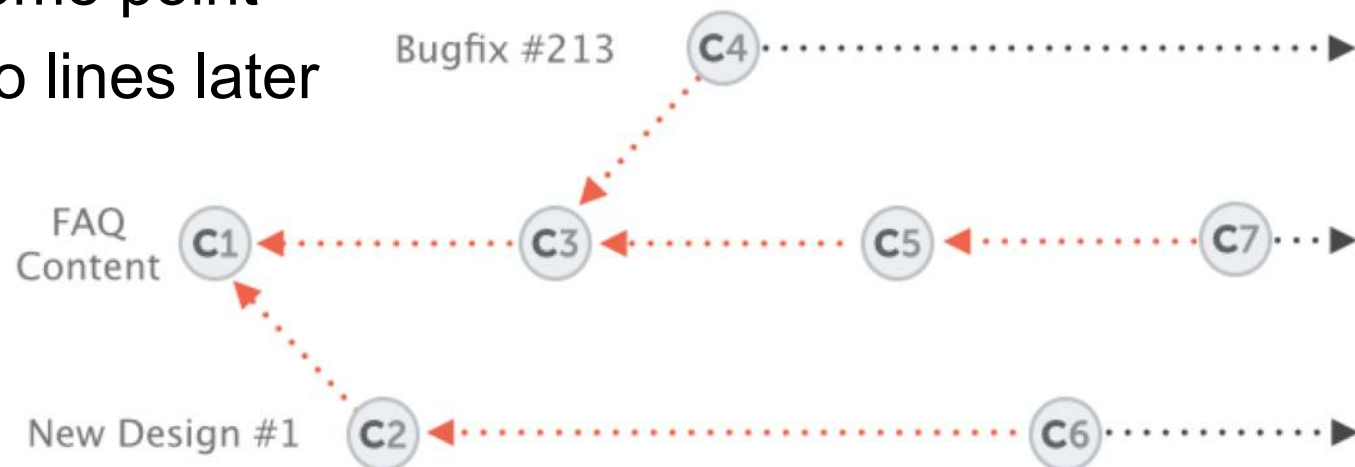
- Connect to remote repo
  - `git remote add origin https://someserver.com/some-repo.git`
- Remote has a name (`origin`) and url
  - Arbitrary name but `origin` is a convention

# Git: Syncing (git push)

- Sync current state to remote with
  - `git push origin master`
  - Pushes the current branch *master* to branch *master* on remote *origin*

## ❖ Branches (ignore for now)

- Independent line of work
- Fork at some point
- Merge two lines later

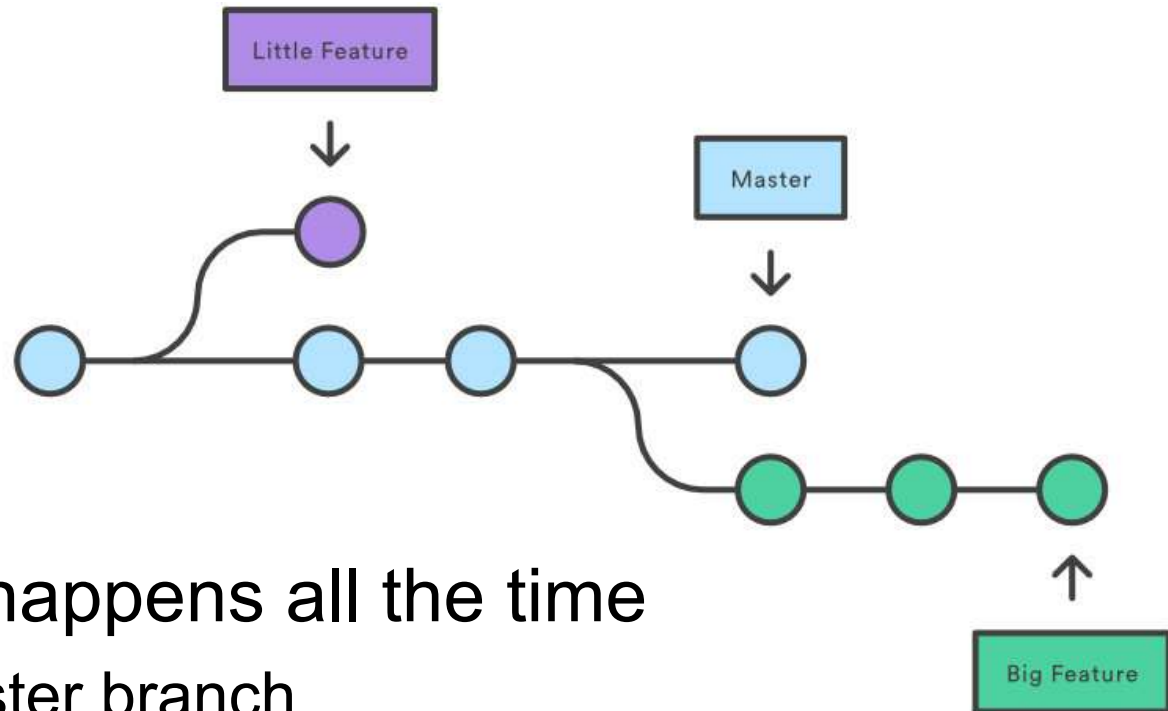


# Git: Syncing (git pull)

- Download and merge changes from remote repo
  - `git pull origin`
- Shortcut for two steps:
  - Fetches changes from remote repo named origin
    - `git fetch origin`
  - Merges the remote branch master into the local branch
    - `git merge origin/master`
- Homework: Continue exercise on Katakoda with scenario 3



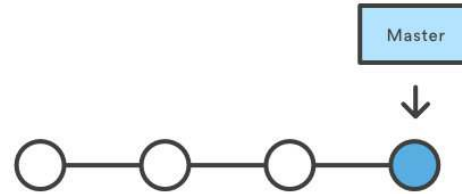
# Git: Branching



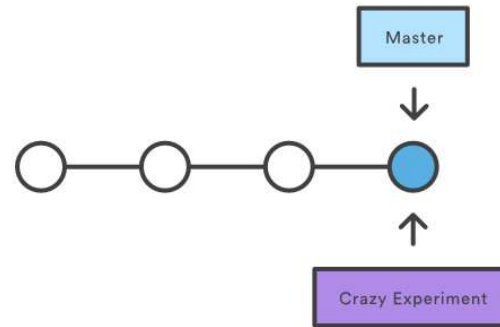
- Branching happens all the time
  - Local master branch
  - Remote master branch
  - Collaborators local master branch
- On push / pull branches get merged

# Git: Branching (git branch)

- On default branch *master*



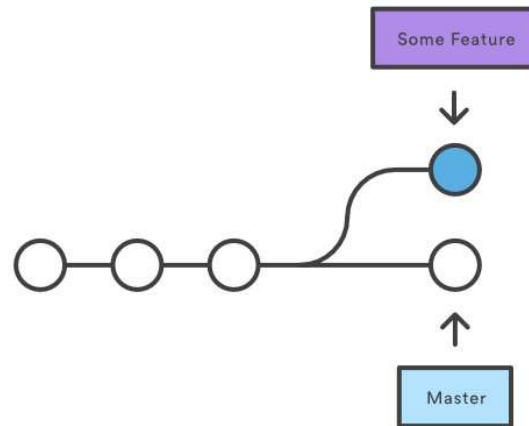
- Creating a branch with `git branch <name>`



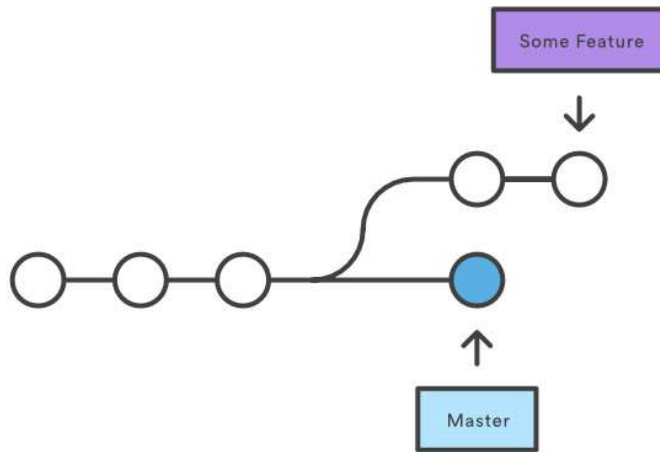
- Nothing happened...

# Git: Branching (git checkout)

- `git checkout <branchname>`
- „Activates“ a branch as the current
  - Updates local files to match the state of the branch
  - All future commits now go into this branch

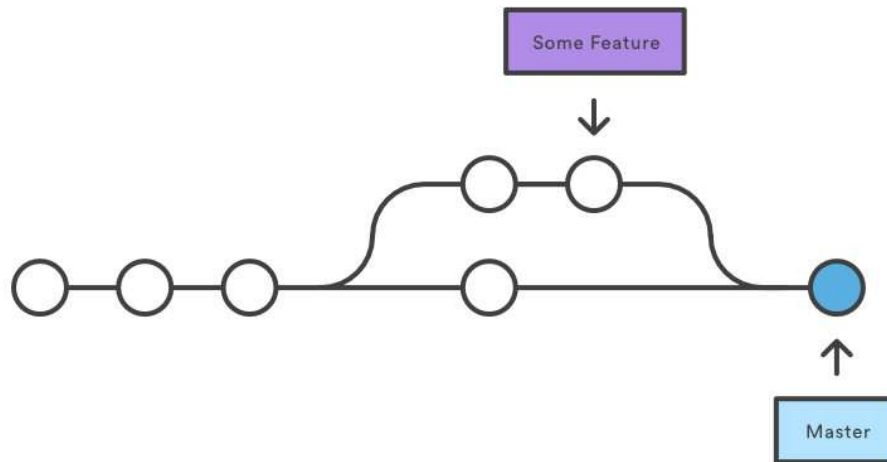


# Git: Branching (git merge)



- Change back to master and merge commits from another into the current (*master*) branch:
  - `git checkout master`
  - `git merge <branchname>`

# Git: Branching (git merge)



- New merge commit is created
- Merged branch can be deleted:
  - `git branch -d <branchname>`
- Homework: Continue exercise on Katakoda until scenario 6

# Git: Conclusion

## ■ Important commands

- `git init <directory> / git clone <url>`
- `git add <file>`
- `git status`
- `git commit -m "..."`
- `git push <remote> <branch>`
- `git pull <remote>`

## ■ Tutorials

- [rogerdudler.github.io/git-guide](https://rogerdudler.github.io/git-guide)
- [git-scm.com/book](https://git-scm.com/book)
- [www.git-tower.com/learn/git/ebook/en/command-line](https://www.git-tower.com/learn/git/ebook/en/command-line)

# Git: Conclusion

- Git enables you to
  - collaborate with arbitrary number of people
  - never lose something
  - have the whole history available
- on any text-based project, e.g.:
  - Paper / reports / books
  - Computer scripts (R, Matlab, ...)
  - Plain text data sets (CSV, GeoJSON, XML-formats, ...)
  - HTML web sites

# Git: Hosting services

- Setup own server
- GitHub – [github.com](https://github.com) free, max. 3 people
- GitLab – [gitlab.com](https://gitlab.com) free
- Bitbucket – [bitbucket.org](https://bitbucket.org) free, max. 5 people



# Git: Hosting services

- Host repositories (download, backup)
- Graphical interface for git
  - Create, modify, delete files
  - History views
  - Diff views
  - Blame views

The screenshot shows a GitHub repository page for 'envinf / OGS-Tutorial-ShallowGeothermal'. The repository is private and has 4 watchers, 0 stars, and 0 forks. The page displays the repository's history, showing 11 commits, 1 branch, 0 releases, and 1 contributor. The latest commit is by 'asachse' and is titled 'The BHE Meshing Tool chapter was implemented as chapter 3. This cause...'. The commit history is as follows:

Commit	Message	Time
chapter01	Tutorial was enhanced by Philipps tutorial material regarding benchma...	11 days ago
chapter02	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
chapter03	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
chapter04	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
chapter05	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
chapter06	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
figures	The BHE Meshing Tool chapter was implemented as chapter 3. This cause...	23 hours ago
.gitattributes	Added .gitattributes & .gitignore files	a month ago
.gitignore	Tutorial was enhanced by Philipps tutorial material regarding benchma...	11 days ago
CLEAR.BAT	upload tutorial	a month ago
README.txt	upload tutorial	a month ago

The right sidebar contains links for Code, Issues (0), Pull requests (0), Wiki, Pulse, Graphs, and Settings. At the bottom, there are options to clone the repository via HTTPS, SSH, or Subversion, and buttons for 'Clone in Desktop' and 'Download ZIP'.

# Git: Hosting services

- Issue tracker (TODOs, discussions)
- Code reviews (Pull requests)
  - Set of commits author wants to integrate in remote repo
  - View changes / comment on them
  - Collaborators review, author reiterates
  - Pull request gets merged
- Project management
  - Milestones (with deadlines, responsible people, issues)

# Exercise: Get example files from GitHub

- <https://github.com/envinf/Hydroinformatik-II>

- Per Git:

```
git clone https://github.com/envinf/Hydroinformatik-II
```

- Or as ZIP

envinf / Hydroinformatik-II

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Professur für Angewandte Umweltsystemanalyse an der TU Dresden, Übungen BHYWI 08 <https://www.ufz.de/index.php?de=40425> Edit

Manage topics

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

bilke Initial.

BHYWI-08-01-E	Initial.	Git URL
BHYWI-08-02-E	Initial.	
BHYWI-08-03-E	Initial.	
BHYWI-08-04-E	Initial.	
BHYWI-08-05-E	Initial.	

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/envinf/Hydroinform>

Open in Desktop Download ZIP

2 days ago

# Exercise / HW: Qt Creator installation

- Download: <https://www.qt.io/offline-installers>
- (OR USB Stick)
- Installation
  - Skip login
  - Qt 5.12.2 (Windows / macOS / Linux)
  - Qt Creator 4.8.2
- Open exercise in Qt Creator
  - Open project -> BHYWI-08-01-E.pro -> Click configure
  - Build executable by clicking green arrow button

# Exercise / HW: Qt Creator installation

The screenshot shows the Qt Creator IDE interface. The left sidebar contains navigation icons for 'Willkommen', 'Editieren', 'Design', 'Debug', 'Projekte', and 'Hilfe'. The 'Projekte' view shows a project named 'BHYWI-08-01-E [master]' with a file 'main.cpp' selected. The main editor displays the following C++ code:

```
1 #include <QApplication>
2 #include <QLabel>
3 int main(int argc, char *argv[])
4 {
5     QApplication app(argc, argv);
6     QLabel *label = new QLabel("<h1><i>Hello</i>"<font color=green>Qt!</font></h1>");
7     label->show();
8     return app.exec();
9 }
10
11
```

Annotations in red text and arrows indicate the following steps:

- 1. Build**: Points to the green play button in the bottom-left toolbar.
- 2. Exe starts**: Points to a window icon displaying 'HelloQt!' in black text.
- 3. Open code**: Points to the 'main.cpp' file in the project view.
- 4. Change color**: Points to the line of code: `QLabel *label = new QLabel("<h1><i>Hello</i>"<font color=green>Qt!</font></h1>");`
- 6. Build again**: Points to the green play button in the bottom-left toolbar.
- 7. New exe**: Points to a second window icon displaying 'HelloQt!' in green text.