= Basiswissen Version Control Systems

git auf der Kommandozeile

Fachinformatiker AE und SI - Version 20230911

*“Destiny, quite often, is a determined parent. Mozart was hardly some naive prodigy who sat down at the keyboard and, with God whispering in his ears, let music flow from his fingertips. It's a nice image for selling tickets to movies, but whether or not God has kissed your brow, you still have to work. Without learning and preparation, you won't know how to harness the power of that kiss.”*

― Twyla Tharp, [The Creative Habit: Learn It and Use It for Life](https://www.goodreads.com/work/quotes/246926)

<https://www.goodreads.com/author/quotes/148884.Twyla_Tharp>

<https://de.wikipedia.org/wiki/Twyla_Tharp>

# Protokoll

MO 4.9. diff # HTML fruits.html

DI 5.9. githowto: 0 - 10 # mini git projekt einrichten

MI 6.9. githowto: 11 - 22 # use checkout to revert changes

DO 7.9. githowto: 23 - 30 # merge conflict

FR 8.9. githowto: 31 - 35, Wiederholung SSH, Accout github # --

MO 11.9. Wiederholung merge, rebase, merge conflict, githowto 36 - 45 # fetch und pull story1

DI 12.9. githowto 46 - 49 Account github 2FA, SSH PubKey # ….

MI 13.9.

DO 14.9.

FR 15.9.

<https://esoteric.codes/blog/suicide-linux>

# == Über dieses Dokument

Dieses Dokument dient als lose Sammlung und Dokumentation der Kursinhalte. Eine eigene Mitschrift kann es (noch) nicht ersetzen.

Disclaimer: Die Konfigurationsbeispiele in diesem Dokument sind zum Lernen gemacht. Zur Verwendung in Produktivumgebungen sind sie nicht (immer) geeignet. Das Dokument wird sich im Laufe des Kurses noch ändern. Alle Beispiele wurden unter Debian GNU/Linux 12 Bookworm getestet.

[source, bash, linenums]

----

**$ git --version**

git version 2.39.2 # Debian 12 bookworm

----

Sie finden das Dokument online hier (bitte machen sie sich ein Bookmark):

[**https://bit.ly/43Ic1hn**](https://bit.ly/43Ic1hn)

# == Über den Dozenten

Werner Heuser studierte Verwaltungswissenschaft an der Verwaltungsfachhochschule Duisburg. Nach Abschluss des Studiums arbeitete er im Kulturamt der Stadt Duisburg und in der Verwaltung verschiedener Krankenhäuser in Berlin. Es folgten einige Jahre als Softwareberater für Personalabrechnungssysteme. 1997 startete er ein Webportal mit Informationen zu Linux mit mobilen Computern. Ab 1999 verkaufte er Laptops, Notebooks und PDAs mit vorinstalliertem Linux. Seit 2015 arbeitet er als freiberuflicher Dozent für Themen aus dem Bereich Linux. Sein Spezialgebiet sind Kurse zur Vorbereitung auf die Prüfungen des Linux-Professional-Institute (LPI).

# == Konzept: Warum git? Welche Zielgruppen? Kursziele

## === Warum git für Anwendungsentwickler?

Historie der Entwicklung (Rollback). Zeitmaschine (time machine).

Softwareentwicklung im Team (collaboration).

## === Warum git für Systemintegratoren?

Verwaltung von Konfigurationsdateien (ansible, docker, documentation, …).

*The* ***etckeeper*** *program is a tool to let /etc be stored in a git, mercurial,*

*brz or darcs repository. It hooks into APT to automatically commit changes*

*made to /etc during package upgrades. It tracks file metadata that version*

*control systems do not normally support, but that is important for /etc, such*

*as the permissions of /etc/shadow. It's quite modular and configurable, while*

*also being simple to use if you understand the basics of working with version*

*control.*

**$ apt show etckeeper**

<https://ubuntu.com/server/docs/tools-etckeeper>

## === Wie ist git entstanden?

*Durch eine Lizenzänderung des bis dahin genutzten proprietären* [*BitKeeper*](https://de.wikipedia.org/wiki/BitKeeper)*-Systems konnten die* [*Linux*](https://de.wikipedia.org/wiki/Linux)*-*[*Kernel*](https://de.wikipedia.org/wiki/Kernel_(Betriebssystem))*-Entwickler dieses nicht mehr kostenlos verwenden, und somit blieb vielen Entwicklern der Zugang verwehrt. Daher begann Torvalds im April 2005 mit der Entwicklung einer neuen Quellcode-Management-Software und präsentierte bereits wenige Tage nach deren Ankündigung eine erste Version.*

<https://de.wikipedia.org/wiki/Git>

## === Warum Linux als Basis für diesen Kurs?

git wurde zunächst für Linuxentwicklung geplant. git läuft zuverlässiger unter Linux. git wird in der Praxis meistens unter Linux verwendet.

<https://www.statista.com/statistics/1296719/tech-skills-that-developers-would-like-to-acquire/>

<https://www.statista.com/statistics/1292382/popular-technologies-in-the-devops-tech-stack/>

## === Warum auf der Kommandozeile?

<https://zwischenzugs.com/2022/04/05/five-reasons-to-master-git-on-the-command-line/> \*\*\*

Bei längeren Kursen wird am Ende des Kurses neben der Kommandozeile VisualStudio Code verwendet.

## === Ziele des Kurses

Teilnehmer sollen einfache (tägliche) Aufgaben mit git auf der Kommandozeile beherrschen (add, commit, clone, ..). Und für die gemeinsame Arbeit im Team gerüstet sein. Gemeinsames arbeiten an einem Projekt in kleinen Gruppen, zu zweit oder dritt. Beispiel: fünf Zeilen HTML und CSS. Github bzw. GitLab und nutzen.

## === Nebenziele des Kurses

Wiederholung und Vertiefung Linux, Wiederholung und Vertiefung HTML und CSS (das könnte aber auch node.js, PHP, JavaScript, C, Python, Java sein).

# == Kursplan

## === Block 1 Grundlagen von git auf der Kommandozeile

### ==== Intro

1. Vorstellung des Dozenten und der Teilnehmer
2. Was macht Versionsverwaltung? Zeitmaschine für (textbasierte) Dateien.
3. Wer hat schon Versionsverwaltung genutzt?
4. Wer kennt HTML?
5. Vorstellung des Unterrichtsplans: Was machen wir an welchem Tag?
6. Welche Dokumentation gibt es?

### ==== Wiederholung Linux

Installation Debian GNU/Linux in einer virtuellen Maschine (VirtualBox)

vi(m), ssh, ls, cd, mkdir, history, TAB completion, ..

Installation zsh plus [Oh-My-Zsh](https://ohmyz.sh/) (OMZ) und Plugins and Themes.

[source, bash, linenums]

----

**# apt update && apt install zsh**

----

[source, bash, linenums]

----

$ omz plugin enable zsh-syntax-highlighting zsh-autosuggestions git git-prompt z colored-man-pages

$ omz theme set bira

----

In $ZSH\_CUSTOM die folgenden Plugins per git clone downloaden:

<https://github.com/zsh-users/zsh-syntax-highlighting>

<https://github.com/zsh-users/zsh-autosuggestions>

Das Ganze mit omz wiederholen für den Standardbenutzer (alice).

Konfiguration vim

[source, bash, linenums]

.~/.vimrc

----

**syntax on**

**autocmd filetype html set shiftwidth=2**

**autocmd filetype css set shiftwidth=2**

**set smartindent**

**" filetype indent on**

----

Hinweis: die gelben Bereiche sind [asciidoc](https://de.wikipedia.org/wiki/AsciiDoc). Bitte nicht mitschreiben! Der Punkt ganz links am Anfang des Dateinamens ist eine Anweisung für asciidoc. Bitte nicht mitschreiben!

Konfiguration nano

[source, bash, linenums]

.~/.nanorc

----

**set linenumbers**

**include "/usr/share/nano/\*.nanorc"**

----

### ==== Das Kommando diff

0 Vorarbeiten (Einstellungen in ~/.bashrc bzw. ~/.zshrc )

Farbige Darstellung von diff

[source, bash, linenums]

.~/.bashrc

----

**alias diff='diff --color=auto'**

----

[source, bash, linenums]

----

**$ source ~/.bashrc**

**$ alias diff**

----

[source, bash, linenums]

----

**$ mkdir -p ~/git\_kurs/diff\_command**

**$ cd ~/git\_kurs/diff\_command**

----

1 Übungsbeispiel zum Kommando diff: Erstellung einer Datei in zwei Versionen

Das Kommando diff ist nicht prüfungsrelevant (für LPIC 701). Aber gut zum Verständnis von git.

Version 1

[source, bash, linenums]

.fruits\_v1.txt

----

**Ananas**

**Aepfel**

**Kiwis**

**Moehren**

**Orangen**

----

Was ist hier in der o.g. Datei falsch?

[source, bash, linenums]

----

**$ cp fruits\_v1.txt fruits\_v2.txt**

----

Version 2

[source, bash, linenums]

**.fruits\_v2.txt**

----

**Ananas**

**Aepfel**

**Kiwis**

**Orangen**

**Grapefruits**

----

2 diff -u zeigt die Unterschiede zwischen den beiden Versionen.

[source, bash, linenums]

----

**$ diff -u fruits\_v1.txt fruits\_v2.txt**

--- fruits\_v1.txt 2022-05-15 08:43:18.813820023 +0200

+++ fruits\_v2.txt 2022-05-15 08:44:36.786387921 +0200

@@ -1,5 +1,5 @@

Ananas

Aepfel

Kiwis

-Moehren

Orangen

+Grapefruits

----

-1,5 erste Datei ab Zeile 1 weitere 5 Zeilen einschliesslich Zeile 1.

+1,5 zweite Datei ab Zeile 1 weitere 5 Zeilen einschliesslich Zeile 1.

Der Befehl `diff -u fruits\_v1.txt fruits\_v2.txt` zeigt was mit Datei `fruits\_v1.txt` passieren muss, damit sie wie Datei `fruits\_v2.txt` aussieht.

Hinweis: die Bedeutung der Option -u ist für das Verständnis nicht wichtig. Wer möchte kann als Alternative zu -u mal -y ausprobieren.

Dokumentation -u

<https://www.gnu.org/software/diffutils/manual/html_node/Unified-Format.html>

### ==== Vergleich einer grösseren Datei

[source, bash, linenums]

----

**$ cp fruits\_v2.txt fruits\_v3.txt**

----

Version 3

[source, bash, linenums]

.fruits\_v3.txt

----

**Ananas**

**Aepfel**

**Kiwis**

**Orangen**

**Grapefruits**

Erdbeeren

Mangos

Ananas

Pflaumen

----

[source, bash, linenums]

----

**$ diff -u fruits\_v2.txt fruits\_v3.txt**

**--- fruits\_v2.txt 2022-05-15 08:44:36.786387921 +0200**

**+++ fruits\_v3.txt 2022-05-16 19:22:57.427004162 +0200**

**@@ -3,3 +3,6 @@ # <1>**

**Kiwis # <2>**

**Orangen**

**Grapefruits**

**+Erdbeeren**

**+Mangos**

**+Pflaumen # <3>**

----

<1> **@@ -3,3 +3,6 @@**

Zeigt die Startzeile an und zählt ab \_und\_ einschliesslich dieser Zeile, z.B. werden aus fruits\_v2.txt die Zeilen -3,3 d.h. 3, 4, 5 angezeigt.

Und aus fruits\_v3.txt werden die Zeilen +3,6 d.h. 3 bis 8 angezeigt.

<2> Das ist Zeile 3 in Version 2 (und in Version 1)

<3> Das ist Zeile 8 in Version 2

Was fällt auf? Zeilen 1 und 2 sind nicht enthalten. Das heisst die Verwaltung der Änderungen nimmt weniger Platz weg, als Kopien der Dateien. Bei diesem kleinen Beispiel macht das noch keinen grossen Effekt. Aber bei grösseren Dateien ist dieser Effekt deutlich merkbar.

Zum besseren Verständnis, die beiden Dateien mit den "original" Zeilennummern.

[source, bash, linenums]

----

**$ diff -u fruits\_v2.txt fruits\_v3.txt | nl --starting-line-number=-2**

----

Oder besser die Dateien einzeln mit nl anzeigen bzw. cat -n?

Auch die ersten drei Zeilen könnten gespart werden. Sie dienen der Darstellung des Kontexts der Änderungen zur besseren menschlichen Lesbarkeit. Mit der Option -U NUM kann die Anzahl der Zeilen mit Kontext verändert werden (default 3).

[source, bash, linenums]

----

**$ diff -U 5 fruits\_v2.txt fruits\_v3.txt**

----

Hinweis: später werden wir git diff benutzen. Das hat auch -U NUM als Option.

Dokumentation

[source, bash, linenums]

----

**$ man diff**

----

### ==== Übungsaufgaben

**Übungsaufgabe**

Bitte kopieren sie **fruits\_v3.txt** nach **fruits\_v4.txt.** Fügen sie die folgenden Zeilen am Ende von **fruits\_v4.txt** hinzu. Ausserdem verändern sie in Zeile 3 Kiwis zu Kibis. Ja das ist ein falsches Wort.

Vergleichen sie dann **fruits\_v3.txt** und **fruits\_v4.txt.**

[source, bash, linenums]

----

**Birnen**

**Pfirsiche**

**Aprikosen**

----

Ändern sie die Reihenfolge der Dateien und vergleichen sie **fruits\_v4.txt** mit **fruits\_v3.txt.**

Zeitrahmen: 20 Minuten

Ein Spezialfall für **diff** ist eine frische, neue Datei, noch ohne Vorgänger. Die Datei wird also aus dem "Nichts" geschaffen. Dieses "Nichts" kann durch die spezielle Datei **/dev/null** simuliert werden.

[source, bash, linenums]

----

**$ diff -u /dev/null fruits\_v1.txt**

--- /dev/null 2023-08-03 06:02:36.996000206 +0200

+++ fruits\_v1.txt 2022-05-15 08:43:18.813820023 +0200

@@ -0,0 +1,5 @@

+Ananas

+Aepfel

+Kiwis

+Moehren

+Orangen

----

Den Inhalt von zwei Verzeichnissen vergleichen.

[source, bash, linenums]

----

**$ cd ~/git\_kurs/**

**$ mkdir diff\_dir**

**$ cd diff\_dir**

**$ mkdir v1 v2**

$ # create a new file v1/fruits.txt with your favorite editor and check it

**$ cat v1/fruits.txt**

Ananas

**$ cp v1/fruits.txt v2/**

$ # create a new file v2/fruits.txt with your favorite editor and check it

**$ cat v2/fruits.txt**

Ananas

Banane

----

[source, bash, linenums]

----

**$ diff -r -u v1 v2**

diff --color -r -u v1/fruits.txt v2/fruits.txt

--- v1/fruits.txt 2023-08-05 06:05:00.880622513 +0200

+++ v2/fruits.txt 2023-08-05 06:06:19.805074532 +0200

@@ -1 +1,2 @@

Ananas

+Banane

----

Hinweis: werden neue Dateien im Verzeichnis v2 angelegt, braucht diff die Option -N.

Git nutzt [SHA1](https://de.wikipedia.org/wiki/Secure_Hash_Algorithm) als "Namen" für seine Objekte.

[source, bash, linenums]

----

**$ diff -r -u v1 v2 | sha1sum**

4411fbaec2d2e2c1ba3bb5436381fa3bfc13c29e -

----

TODO hier besser Dateien mit diff vergleichen.

[source, bash, linenums]

----

**$ sha1sum fruits\_v1.txt**

78553e01e36735fb54e4da0deebee4ac0fd91c0b fruits\_v1.txt

----

[source, bash, linenums]

----

**$ diff -u fruits\_v1.txt fruits\_v2.txt | sha1sum**

3b026b77e9e19b8bdcc8840c9c49261861508fbe -

----

**Fazit**

diff und auch git arbeiten mit **Textdateien** (d.h. Bilder, Audiodateien und Filme sind (erstmal) nicht im Fokus. Die Differenzen zwischen den verschiedenen Textdateien werden zeilenweise verarbeitet (das Verfahren kennt man von grep). Das heisst, auch wenn in einer Zeile nur ein einzelnes Zeichen verändert wurde, gilt die ganze Zeile als verändert.

**Ausblick**

Warum haben wir uns mit diff beschäftigt. git verwaltet Änderungen an Objekten. Dabei speichert es nur die Differenzen (nun ja das stimmt wohl nicht so ganz, trotzdem ist es erstmal für den Anfang eine erste Annäherung an die Arbeitsweise von git). Wir werden diese Differenzen später mit git diff anschauen.

### ==== HTML & CSS Basics

Wir legen uns ein Arbeitsverzeichnis an.

[source, bash, linenums]

----

**$ mkdir -p ~/git\_kurs/html\_basics**

**$ cd ~/git\_kurs/html\_basics**

----

Eine kleine HTML Datei.

[source, html, linenums]

.fruits.html

----

**<!DOCTYPE html>**

**<html lang="de">**

**<head>**

**<title>Obstladen</title>**

**<link href="style.css" rel="stylesheet">**

**</head>**

**<body>**

**<h1>Hier gibt es das beste Obst!</h1>**

**<p>**

**Coole Früchte und mehr.**

**</p>**

**</body>**

**</html>**

----

<https://wiki.selfhtml.org/wiki/HTML/Tutorials/Grundger%C3%BCst>

Eine kleine CSS Datei.

[source, css, linenums]

.style.css

----

**p {**

**color: purple;**

**font-size: larger;**

**}**

----

Hinweis: Wir brauchen **kleine HTML und CSS Dateien**, um den Überblick zu behalten. Das müssen keine perfekten Dokumente sein.

Wie können wir unser Projekt anschauen?

A Start eines Browsers über die Kommandozeile

[source, bash, linenums]

----

**$ firefox fruits.html &**

**$ xdg-open fruits.html**

----

B Im Browser lokal

**file:///home/alice/git\_kurs/html\_basics/fruits.html**

C Im Browser über einen Webserver

Installation eines Webservers hier nginx. Das könnte aber auch ein anderer Webserver sein.

[source, bash, linenums]

----

**# apt update && apt install nginx**

----

Der Webserver findet seine Dateien per Default unter **/var/www/html/** Wir verlinken unser Arbeitsverzeichnis dorthin.

[source, bash, linenums]

----

**# ln -s /home/alice/git\_kurs/ /var/www/html/**

----

Nun können wir im Browser die IP Adresse (oder den Namen) unseres Webservers eingeben. Zum Beispiel

[source, bash, linenums]

----

[**http://localhost/git\_kurs/html\_basics/fruits.html**](http://localhost/git_kurs/html_basics/fruits.html) **# Umlaut "ü" falsch**

[**http://localhost/git\_kurs/html\_basics/fruits\_v2.html**](http://localhost/git_kurs/html_basics/fruits_v2.html) **# Umlaut "ü" korrekt**

----

Oder vielleicht besser in

[source, bash, linenums]

----

**/etc/nginx/site-enabled/default**

**autoindex on;**

----

<https://www.keycdn.com/support/nginx-directory-index>

### ==== HTML & CSS Validation

In Produktionsumgebungen sollte immer eine Validierung der Objekte erfolgen. Dazu installieren wir uns zwei Validierungstools. Ein Tool für HTML und ein Tool für CSS.

[source, bash, linenums]

----

**# apt update && apt install tidy csstidy**

----

[source, bash, linenums]

----

**$ apt show tidy**

*[..]*

*Tidy corrects and cleans up HTML and XML documents by fixing*

*markup errors and upgrading legacy code to modern standards.*

----

[source, bash, linenums]

----

**$ apt show csstidy**

*[..]*

*csstidy parses and optimises your CSS code, making it cleaner and more*

*concise. The end result is a smaller CSS file and better written code. It*

*has a variety of option settings giving the user a significant amount of*

*control over the level of file compression and readability. It is not meant*

*to be a CSS validator.*

----

<https://www.smashingmagazine.com/2016/05/stylelint-the-style-sheet-linter-weve-always-wanted/>

<http://csstidy.sourceforge.net/>

<https://stackoverflow.com/questions/6646204/are-there-any-command-line-validation-tools-for-html-and-css>

[source, bash, linenums]

----

**$ tidy -e fruits.html**

**$ csstidy style.css**

----

**Übungsaufgabe**

Bitte machen sie sich eine Kopie der Datei fruits.html Die Kopie soll heissen fruits\_v2.html. Fügen sie bitte in der Kopie die folgende neue Zeile hinzu. Wo muss diese Zeile hin? Bitte überprüfen sie mit tidy. Und nutzen sie diff um den Unterschied zwischen den beiden Dateien darzustellen.

[source, html, linenums]

.fruits\_v2.html

----

**<meta charset="utf-8">**

----

Zeitrahmen: 20 Minuten bis 14:45

Fazit

Warum nutzen wir HTML und CSS? Wir brauchen ein möglichst einfaches (und für Alle verständliches) Beispiel zum Kennenlernen von git. Selbstverständlich hätten wir auch reinen Text, C, Python, Javascript, eine Konfigurationsdatei (z.B. ~/.zshrc) etc. nehmen können.

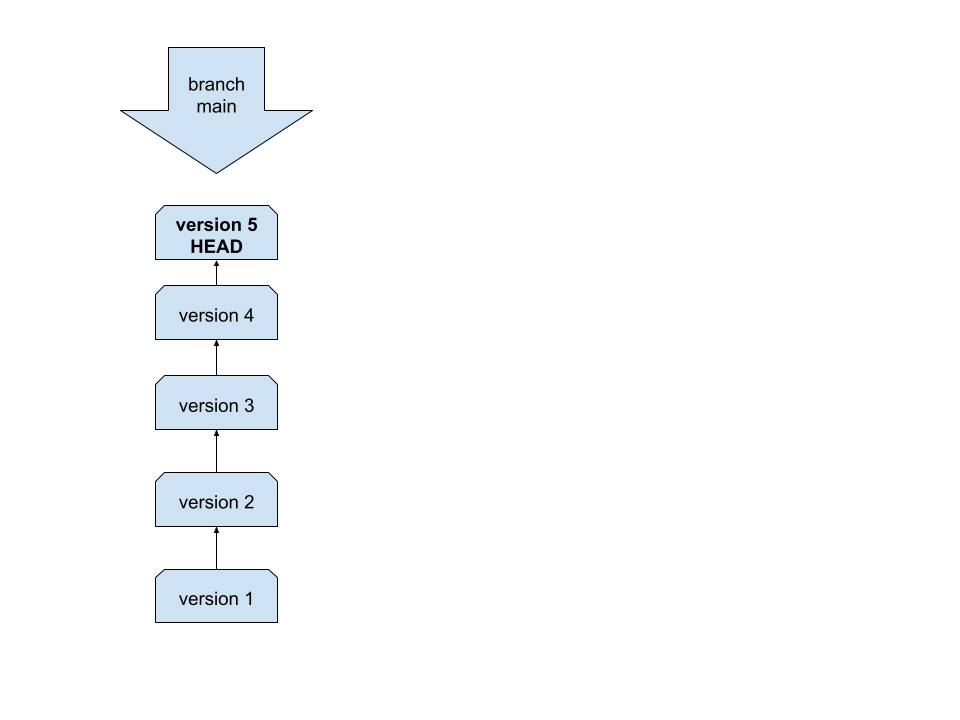
Warum nutzen wir tidy und csstidy? git wird in der Praxis zusammen mit Continous Integration – CI / Continous Delivery – CD verwendet. Mit tidy und csstidy können wir später ein entsprechendes Beispiel entwickeln. Oder ein Beispiel mit git hooks.

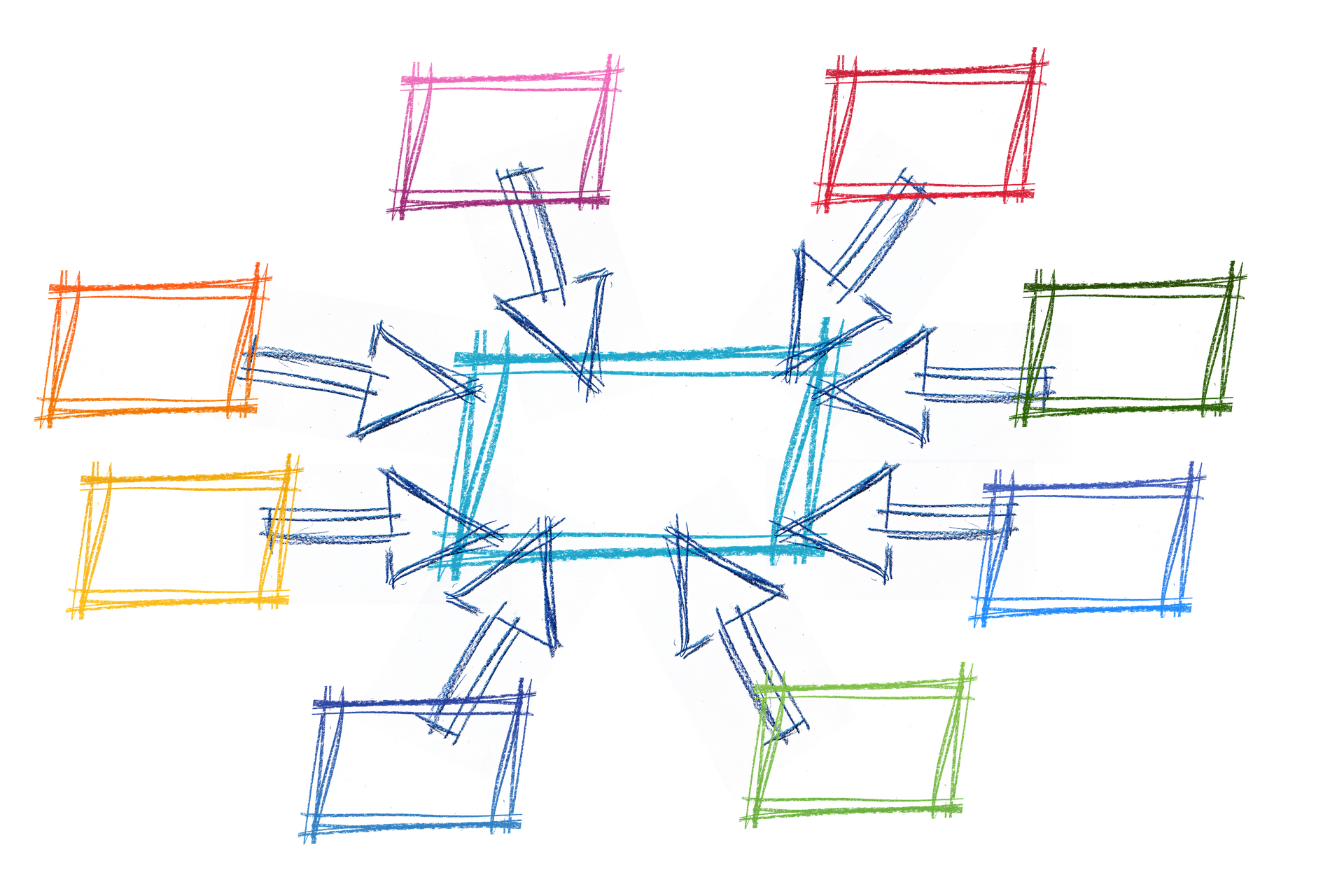
### ==== Was ist Versionskontrolle?

*"Version tools are not just important for maintaining a history of a project, they are also the foundation for a team to collaborate."*

source: <https://martinfowler.com/bliki/VersionControlTools.html>

**Version History (time line)**



**Zusammenarbeit (Collaboration)** 

<https://pixabay.com/de/illustrations/netzwerk-rechteck-ringe-vernetzen-1989146/>

***Git*** *(*[*/ɡɪt/*](https://en.wikipedia.org/wiki/Help:IPA/English)*)*[*[8]*](https://en.wikipedia.org/wiki/Git#cite_note-:0-9) *is a* [*distributed version control*](https://en.wikipedia.org/wiki/Distributed_version_control) *system that tracks changes in any set of* [*computer files*](https://en.wikipedia.org/wiki/Computer_file)*, usually used for coordinating work among* [*programmers*](https://en.wikipedia.org/wiki/Programmer) *collaboratively developing* [*source code*](https://en.wikipedia.org/wiki/Source_code) *during* [*software development*](https://en.wikipedia.org/wiki/Software_development)*. Its goals include speed,* [*data integrity*](https://en.wikipedia.org/wiki/Data_integrity)*, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).*[*[9]*](https://en.wikipedia.org/wiki/Git#cite_note-kernel_SCM_saga-10)[*[10]*](https://en.wikipedia.org/wiki/Git#cite_note-integrity_goals-11)[*[11]*](https://en.wikipedia.org/wiki/Git#cite_note-linusGoogleTalk-12)

*Git was originally authored by* [*Linus Torvalds*](https://en.wikipedia.org/wiki/Linus_Torvalds) *in 2005 for development of the* [*Linux kernel*](https://en.wikipedia.org/wiki/Linux_kernel)*, with other kernel developers contributing to its initial development.*[*[12]*](https://en.wikipedia.org/wiki/Git#cite_note-pro-git-1.2-13) *Since 2005, Junio Hamano has been the core maintainer. As with most other* [*distributed version control*](https://en.wikipedia.org/wiki/Distributed_version_control) *systems, and unlike most* [*client–server*](https://en.wikipedia.org/wiki/Client%E2%80%93server) *systems, every Git* [*directory*](https://en.wikipedia.org/wiki/Directory_(computing)) *on every* [*computer*](https://en.wikipedia.org/wiki/Node_(networking)) *is a full-fledged* [*repository*](https://en.wikipedia.org/wiki/Repository_(version_control)) *with complete history and full version-tracking abilities, independent of network access or a central server.*[*[13]*](https://en.wikipedia.org/wiki/Git#cite_note-14) *Git is* [*free and open-source software*](https://en.wikipedia.org/wiki/Free_and_open-source_software) *distributed under the* [*GPL-2.0-only*](https://en.wikipedia.org/wiki/GNU_General_Public_License) *license.*

<https://en.wikipedia.org/wiki/Git> \*\*\* recommended read

Revision control systems are often centralized, with a single authoritative data store, the *repository,* and check-outs and check-ins done with reference to this central repository. Alternatively, in [distributed revision control](https://en.wikipedia.org/wiki/Distributed_revision_control), no single repository is authoritative, and data can be checked out and checked into any repository. When checking into a different repository, this is interpreted as a merge or patch.

<https://en.wikipedia.org/wiki/Version_control>

## === Preparation: Installation and Global Configuration

### ==== Installation git githowto 0

**Ab hier machen wir mit der Anleitung aus dem githowto weiter. Die Anleitung erklärt alle wichtigen git basics mit einfachen Beispielen. Die Anleitung besteht aus 51, meist einseitigen, Artikeln. Das heisst, der Lernumfang ist überschaubar.**

<https://githowto.com/> (<https://github.com/GitHowTo/githowto-content> )

Installation git unter Debian GNU/Linux

[source, bash, linenums]

----

**# apt update && apt install git**

----

### ==== Global Configuration githowto 1 Preparation

<https://githowto.com/setup>

Minimale git Konfiguration (--global setzt die Angaben für alle zukünftigen git Projekte):

[source, bash, linenums]

----

**$ git config --global user.email "david.damago@gmx.de"**

**$ git config --global user.name "David Damago"**

**$ git config --global core.pager "less -F"**

**# $ git config --global core.editor "vim"**

**$ git config --global core.editor "nano"**

**# automatic CR and LF handling on Unix/Mac**

**$ git config --global core.autocrlf input**

**$ git config --global core.safecrlf warn**

**# default branch is "master"**

**$ git config --global init.defaultbranch "main"**

----

Hinweis: sollen die Angaben geändert werden, die o.g. Befehle einfach nochmal mit den richtigen Angaben ausführen. Oder die Konfigurationsdatei, siehe unten, ändern.

Kontrolle der git Konfiguration:

[source, bash, linenums]

----

**$ git config --global --list**

user.name=David Damago

user.email=david.damago@gmx.de

core.pager=less -F

core.editor=vim

init.defaultbranch=main

# [..]

----

Wo wird die globale git Konfiguration gespeichert?

[source, bash, linenums]

----

**$ cat ~/.gitconfig**

[user]

name = David Damago

email = david.damago@gmx.de

[core]

pager = less -F

editor = vim

[init]

defaultBranch = main

# [..]

----

$ man git-config

<https://manpages.debian.org/bookworm/git-man/git-config.1.en.html>

### ==== Download git howto stuff githowto 2 The final preparation

<https://githowto.com/more_setup>

[source, bash, linenums]

----

**$ cd ~/git\_kurs**

**$ wget https://githowto.com/git\_tutorial.zip**

**$ unzip git\_tutorial.zip**

**$ tree git\_tutorial | less**

----

Wir brauchen den o.g. Download für die folgenden Übungen nur als Backup.

[source, bash, linenums]

----

**$ ls -lA ../files/lab\_\* -d**

drwxr-xr-x 2 alice alice 4096 Apr 10 2011 ../files/lab\_03

drwxr-xr-x 2 alice alice 4096 Aug 1 16:58 ../files/lab\_05

drwxr-xr-x 2 alice alice 4096 Apr 10 2011 ../files/lab\_10

drwxr-xr-x 2 alice alice 4096 Apr 10 2011 ../files/lab\_20

drwxr-xr-x 3 alice alice 4096 Apr 10 2011 ../files/lab\_24

drwxr-xr-x 3 alice alice 4096 Apr 10 2011 ../files/lab\_25

drwxr-xr-x 3 alice alice 4096 Aug 1 09:12 ../files/lab\_30

drwxr-xr-x 3 alice alice 4096 Apr 10 2011 ../files/lab\_35

drwxr-xr-x 3 alice alice 4096 Apr 10 2011 ../files/lab\_45

drwxr-xr-x 3 alice alice 4096 Apr 10 2011 ../files/lab\_50

----

## === Creating a First git Project

### ==== Konfiguration eines ersten git Repository githowto 3 Creating a project

<https://githowto.com/create_a_project>

Anlegen des Arbeitsverzeichnisses ( working directory, das heisst besser: working tree weil hier auch noch Unterverzeichnisse eingerichtet werden können).

[source, bash, linenums]

----

$ cd ~/git\_kurs/git\_tutorial/work

$ mkdir hello

$ cd hello

$ # create a new file hello.html with your favorite editor and check it

$ cat hello.html

Hello, World!

----

Initialisierung des git Repository

[source, bash, linenums]

----

**$ git init**

Initialized empty Git repository in /[..]/git\_kurs/git\_tutorial/work/hello/.git/

----

Wo legt git seine Daten ab? Wo ist das (local) git Repository?

[source, bash, linenums]

----

**$ tree -F .git**

.git

├── branches/

├── config

├── description

├── HEAD

├── hooks/

│ ├── applypatch-msg.sample\*

│ ├── commit-msg.sample\*

[...]

├── info/

│ └── exclude

├── objects/

│ ├── info/

│ └── pack/

└── refs/

├── heads/

└── tags/

9 directories, 17 files

----

**$ man git-init**

<https://git-scm.com/docs/git-init>

<https://git-scm.com/docs/gitrepository-layout>

<https://www.atlassian.com/de/git/tutorials/setting-up-a-repository/git-init> \*\*

Das git Plugin der zsh hat nach der Initialisierung des git Repositories den Prompt geändert:

**╭─alice@werner ~/git\_kurs/git\_tutorial/work ‹main●›**

**╰─$**

Wer unbedingt mit der BaSH arbeiten muss:

<https://github.com/magicmonty/bash-git-prompt>

<https://github.com/git/git/blob/master/contrib/completion/git-completion.bash>

[source, bash, linenums]

----

**$ git add hello.html**

**$ git commit -m "First Commit"**

----

[source, bash, linenums]

----

**$ git add hello.html**

**$ git commit -m "First Commit"**

[master (root-commit) 911e8c9] First Commit

1 files changed, 1 insertions(+), 0 deletions(-)

create mode 100644 hello.html

----

<https://manpages.debian.org/bookworm/git-man/git-add.1.en.html>

<https://manpages.debian.org/bookworm/git-man/git-commit.1.en.html>

### ==== githowto 4 Checking the status of the project

<https://githowto.com/checking_status>

Was weiss git über den Zustand (status) des working tree und des Repository?

[source, bash, linenums]

----

**$ git status**

On branch main

nothing to commit, working tree clean

----

<https://manpages.debian.org/bookworm/git-man/git-status.1.en.html>

### ==== githowto 5 Making changes

<https://githowto.com/making_changes>

[source, bash, linenums]

----

**$ git status --short**

M hello.html

----

[source, bash, linenums]

----

*In the short-format, the status of each path is shown as one of these forms*

**XY PATH**

**XY ORIG\_PATH -> PATH**

*where* **ORIG\_PATH** *is where the renamed/copied contents came from.* **ORIG\_PATH** *is only shown when the entry is renamed or copied. The* **XY** *is a two-letter status code.*

----

<https://manpages.debian.org/bookworm/git-man/git-status.1.en.html>

### ==== githowto 6 Staging the changes

<https://githowto.com/staging_changes>

Hinweis: im githowto wird git reset HEAD <file> genannt, um Änderungen aus dem stage (aka index) zurückzuholen. Die moderne Form aus dem git status Kommando empfiehlt:

[source, bash, linenums]

----

use git restore --staged <file>..." to unstage

----

Hinweis: man kann sich vorher die Unterschiede anschauen:

[source, bash, linenums]

----

**$ git diff --staged <file>**

----

### ==== githowto 7 Staging and commiting [Why make it complicated?]

Dieser Teil des githowto ist dort nur theoretisch ausgeführt. Wir machen das aber als Übung ganz konkret in einem neuen git repository.

[source, bash, linenums]

----

$ cd ~/git\_kurs/

$ mkdir workflow

$ cd workflow

$ touch a.html b.html c.html

----

Initialisierung des git Repository

[source, bash, linenums]

----

**$ git init**

Initialized empty Git repository in /[..]/git\_kurs/workflow/.git/

----

[source, bash, linenums]

----

**$ git add a.html**

**$ git add b.html**

**$ git commit -m "Changes for a and b"**

main (root-commit) 18d740e] Changes for a and b

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 a.html

create mode 100644 b.html

----

[source, bash, linenums]

----

**$ git add c.html**

**$ git commit -m "Unrelated change to c"**

[main 1f902d2] Unrelated change to c

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 c.html

----

[source, bash, linenums]

----

**$ git log --oneline**

1f902d2 (HEAD -> main) Unrelated change to c

18d740e Changes for a and b

----

<https://githowto.com/staging_and_committing>

**Workflow**

1. Wir erstellen (oder ändern) eine HTML Datei im git **working tree**.
2. Wir melden diese Datei im git **stage (aka cache, aka index)** an: git add
3. Wir übertragen die Datei vom git stage in das git **repository (local)**: git commit
4. [Wir übertragen die Datei in ein **repository (remote)**: git push] (das kommt später)

working tree -> stage (aka index) -> repository (local) -> repository (remote z.B. github)

Warum ist das so kompliziert? Warum können die Objekte über den Umweg stage ins Repository? Man kann sich das **Repository** wie ein Materiallager vorstellen. Und den **Stage (aka index)** wie ein [Tablett](https://de.wikipedia.org/wiki/Tablett) mit dem Objekte ins Materiallager transportiert werden.

Siehe auch das Schaubild hier:

<https://www.baeldung.com/git-guide> 7. Git Workflow

**Praxistipp**

Wir machen Änderungen in den Dateien **a.html** und **b.html**. Mit **git add -A** können wir alle geänderten Dateien in den stage aufnehmen. Mit der Option **--dry-run** kann vorher ein Testlauf gemacht werden.

[source, bash, linenums]

----

**$ git add -A --dry-run**

add 'a.html'

add 'b.html'

----

**Praxistipp**

Wir machen Änderungen in den Dateien **a.html** und **b.html**. Mit **git commit -a** können wir das **git add** sparen. Mit der Option **--dry-run** kann vorher ein Testlauf gemacht werden.

[source, bash, linenums]

----

**$ git commit -a --dry-run**

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: a.html

modified: b.html

----

### ==== githowto 8 Committing the changes

<https://githowto.com/commiting_changes>

### ==== Änderungen werden verwaltet, und nicht Dateien githowto 9 Changes, not files

<https://githowto.com/changes_not_files>

*"Git concentrates on the changes to a file, not the file itself. A git add file command does not tell git to add the file to the repository, but to note the current state of the file for it to be commited later."*

Nach den **ersten** Änderungen der Datei **hello.html** d.h. noch vor   
**git add hello.html** erstmal **git status** aber dann :

[source, bash, linenums]

----

**$ git status -v -v**

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

--------------------------------------------------

Changes not staged for commit:

diff --git i/hello.html w/hello.html

index f76d8e9..6da0629 100644

--- i/hello.html

+++ w/hello.html

@@ -1 +1,5 @@

-<h1>Hello, World!</h1>

+<html>

+ <body>

+ <h1>Hello, World!</h1>

+ </body>

+</html>

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

----

-v steht (wie so oft) für verbose -v -v steht für noch mehr verbose

w working tree

i index (aka stage)

**$ git status --short**

**$ git show HEAD** # tab completion shows other revisions

<https://manpages.debian.org/bookworm/git-man/git-show.1.en.html>

Nach den **zweiten** Änderungen der Datei **hello.html** d.h. noch vor dem zweiten  
 **git add hello.html** erstmal **git status** aber dann :

[source, bash, linenums]

----

**$ git status -vv**

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

Changes to be committed:

diff --git c/hello.html i/hello.html

index f76d8e9..6da0629 100644

--- c/hello.html

+++ i/hello.html

@@ -1 +1,5 @@

-<h1>Hello, World!</h1>

+<html>

+ <body>

+ <h1>Hello, World!</h1>

+ </body>

+</html>

--------------------------------------------------

Changes not staged for commit:

diff --git i/hello.html w/hello.html

index 6da0629..0d576c4 100644

--- i/hello.html

+++ w/hello.html

@@ -1,4 +1,6 @@

<html>

+ <head>

+ </head>

<body>

<h1>Hello, World!</h1>

</body>

----

c changes

**$ git status --short**

Am Ende dieses Abschnitts (das kommt aber auch gleich im nächsten Abschnitt):

**$ git log --oneline**

### ==== githowto 10 History

<https://githowto.com/history>

<https://manpages.debian.org/bookworm/git-man/git-log.1.en.html>

Praxistipp

[source, bash, linenums]

----

**$ git log --oneline**

----

### ==== Was haben wir erreicht A? git funktioniert schon das kann man mit git log und git diff sehen

Was ist bisher in unserem git Repository passiert?

git log

git show

git diff

Wir könnten bereits die Änderungen im working tree rückgängig machen. Aus dem stage, aus dem repository.

Zusätzlich zum githowto das Thema mit **git show** und **git diff** erfahrbar machen.

[source, bash, linenums]

----

**$ git show 48b2505:hello.html**

----

[source, bash, linenums]

----

**$ git diff 48b2505:hello.html 2b033fa:hello.html**

diff --git a/hello.html b/hello.html

index 8ab686e..6da0629 100644

--- a/hello.html

+++ b/hello.html

@@ -1 +1,5 @@

-Hello, World!

+<html>

+ <body>

+ <h1>Hello, World!</h1>

+ </body>

+</html>

----

**Übungsaufgabe**

Wir haben git init in unseren Projekt Verzeichnis **~/git\_kurs/git\_tutorial/kurs/hello** ausprobiert. Man kann aber auch bestehende Projekte mit git init zur Versionskontrolle hinzufügen.

Legen sie einen Ordner **~/git\_kurs/my\_first\_git** an. Dann wechseln sie in diesen Ordner und erstellen ein kleines Programm (python, bash, java, html, …). Erst danach führen sie git init aus (diese Reihenfolge ist aber nicht zwingend notwendig, d.h. sie können auch als allererstes den git init machen). Dann überprüfen sie den Status des git Repository und die Logdatei.

Was müssen sie alles machen für den ersten commit ins git Repository? Führen sie den commit durch. Dann überprüfen sie nochmal den Status des git Repository und die Logdatei.

Zeit 30 Minuten

Danach stellen sie ihre Projekte in der Klasse vor.

### ==== Exkurs: Wie findet man die git man pages?

Es gibt etwa 190 man pages zu git.

[source, bash, linenums]

----

**$ man git-config**

**$ man git-init**

**$ man git-add**

**$ man git-commit**

**$ man git-checkout**

**$ man git-restore**

**$ man git-status**

**$ man git-log**

**$ man git-diff**

**$ [..]**

----

<https://manpages.debian.org/bookworm/git-man/index.html>

[source, bash, linenums]

----

**$ git help**

**$ git help -a**

**$ git help -g**

----

<https://manpages.debian.org/bookworm/git-man/git-help.1.en.html>

Wie liest man eine man page?

Kurze Hilfe zu Kommando mit der Option -h

[source, bash, linenums]

----

**$ git commit -h 2>&1 | less**

**# ..**

----

### ==== Änderungen im Detail: working tree -> stage (aka index) -> repository (local)

Weiter unten (oder oben ?) gibt es Schaubilder!

[source, bash, linenums]

----

**$ git ls-files** # working directory

**$ git ls-files --stage** # stage (index)

**$ git ls-tree HEAD** # repository

----

<https://www.atlassian.com/git/tutorials/inspecting-a-repository> \*\*

### ==== githowto 11 Aliases

<https://githowto.com/aliases>

<https://manpages.debian.org/bookworm/git-man/git-config.1.en.html>

Anzeige der git Aliase:

[source, bash, linenums]

----

**$ git config --global --list**

----

Hinweis: Die Shell-Aliase sind bereits per [Oh-My-Zsh](https://ohmyz.sh/) (OMZ) Plugin installiert. Das brauchen wir also nur zu machen, wenn wir die BaSH nutzen. Anzeige der Shell Aliase:

[source, bash, linenums]

----

**$ alias | grep "git" | less**

----

Wer die BaSH nutzen möchte:

<https://think-like-a-git.net/sections/graphs-and-git/visualizing-your-git-repository.html>

<https://github.com/Bash-it/bash-it>

### ==== git checkout - githowto 12 Getting older versions

<https://githowto.com/getting_old_versions>

<https://manpages.debian.org/bookworm/git-man/git-checkout.1.en.html>

Now we are ready for our first time travel back into the past with git.

Bevor wir starten, schauen wir uns die aktuelle Situation an.

[source, bash, linenums]

----

**$ cat hello.html**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

----

Was bedeutet diese Meldung nach dem ersten checkout? Wie hat sich der zsh Prompt verändert?

[source, bash, linenums]

----

[..]

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

[..]

----

[source, bash, linenums]

----

**$ git status**

HEAD detached at 856039e

nothing to commit, working tree clean

----

Hinweis: im zsh Prompt sieht man auch den Hash des commits.

### ==== Was haben wir erreicht B? time travel funktioniert schon: git checkout

Wir haben bereits mehrere commits! Das heisst wir können auch zu früheren commits zurückkehren.

[source, bash, linenums]

----

**$ git checkout 856039e**

**$ cat hello.html**

**$ # ..**

**$ git checkout 237111d**

**$ cat hello.html**

**$ # ..**

**$ git checkout main**

**$ cat hello.html**

**$ # ..**

----

### ==== Tags githowto 13 Tagging versions

<https://githowto.com/tagging_versions>

<https://manpages.debian.org/bookworm/git-man/git-tag.1.en.html>

Bevor wir beginnen, überprüfen wir den aktuellen Zustand. Der sollte so aussehen:

[source, bash, linenums]

----

**$ git status**

On branch main

nothing to commit, working tree clean

**$ git log --oneline --all**

[...]

----

| **SHA-1 hash** | **Beschreibung** | **pointer** | **pointer** | **tag** | **tag** | **misc** |
| --- | --- | --- | --- | --- | --- | --- |
| 4428077 | (HEAD -> main, tag: v1) changes to A B C | HEAD | HEAD | v1 | v1 | @ (aktueller HEAD)  main |
| 5e492ea | (tag: v1-beta) Added standard HTML page tags | HEAD^ | HEAD~1 | v1^ | v1~1 | v1-beta |
| 382ea6a | a in A geändert | HEAD^^ | HEAD~2 | v1^^ | v1~2 |  |
| 936db94 | the release contains c.html | HEAD^^^ | HEAD~3 | v1^^^ | v1~3 |  |

Für weitere Bezeichnungen siehe

<https://www.git-scm.com/docs/gitrevisions> \*\*\* BTW eine der wichtigsten Manualseiten.

<https://stackoverflow.com/questions/15973182/git-why-are-double-dashes-needed-when-running-a-command-on-a-deleted-file>

<https://git-scm.com/book/de/v2/Git-Tools-Revisions-Auswahl> short tutorial

<https://devhints.io/git-revisions> git revisions cheat sheet

Was passiert, wenn wir bei git hist die Option --all weglassen? Der branch wird nur bis zum aktuellen HEAD angezeigt.

Die folgenden Tools nutzen um die Navigation im branch main zu üben. Beispiele

[source, bash, linenums]

----

**$ git checkout v1-beta hello.html**

**$ git diff HEAD:hello.html v1-beta:hello.html**

**$ git show HEAD:hello.html**

**$ git grep -e "h1" v1-beta**

----

Hinweis: die Autovervollständigung der Shell kennt die Bezeichnungen aus der o.g. Tabelle.

<https://manpages.debian.org/bookworm/git-man/gitcli.7.en.html> \*\*\*

<https://zwischenzugs.com/2016/06/04/power-git-log-graphing/> \*\*\*

<https://www.ubuntumint.com/git-commands/#10_git_diff_Tack_Changes_in_Files> more git diff examples

Einen zusätzlichen tag vergeben.

[source, bash, linenums]

----

**$ git tag test\_me\_tag 18d740e**

----

Alle verfügbaren tags anzeigen:

[source, bash, linenums]

----

**$ git tag --list**

----

Einen tag entfernen kommt etwas später.

<https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag>

## === How to Handle Mistakes

### ==== githowto 14 Discarding local changes

<https://githowto.com/undoing_local_changes>

<https://manpages.debian.org/bookworm/git-man/git-checkout.1.en.html>

An Stelle von **git checkout hello.html** kann man die bei   
**git status** empfohlene Form benutzen.

(use "git restore <file>..." to discard changes in working directory)

<https://manpages.debian.org/bookworm/git-man/git-restore.1.en.html>

<https://www.heise.de/developer/meldung/Versionsverwaltung-Git-2-23-Experimentelle-Alternativen-zu-Git-Checkout-4500052.html> \*\*\* Anmerkungen warum restore an Stelle von checkout.

### ==== githowto 15 Cancel Staged changes (before committing)

<https://githowto.com/undoing_staged_changes>

<https://manpages.debian.org/bookworm/git-man/git-reset.1.en.html>

reset versus restore

### ==== githowto 16 Cancelling commits

<https://githowto.com/undoing_committed_changes>

<https://manpages.debian.org/bookworm/git-man/git-revert.1.en.html>

Hinweis: Durch den revert wird der working tree wiederhergestellt UND ein neuer Eintrag in den Logs gemacht! Siehe **git log --online** Das heisst der revert ist ein (spezieller) commit zur Rückgängigmachung früherer Änderungen.

Das githowto sollte um eine Überprüfung des Dateiinhalts nach dem revert mit   
**cat hello.html** ergänzt werden.

Praxistipp:

Die Option **--no-edit** nutzen.

[source, bash, linenums]

----

**$ git revert HEAD --no-edit**

----

### ==== githowto 17 Removing a commit from a branch

<https://githowto.com/removing_commits_from_a_branch>

<https://manpages.debian.org/bookworm/git-man/git-reset.1.en.html>

[source, bash, linenums]

----

**$ git log --oneline**

**$ git log --oneline --all**

----

Zusammenfassung: checkout, reset, revert, [restore not (officially) used in githowto, but added by WH see above].

<https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting> \*\*\* Schaubilder

<https://www.geeksforgeeks.org/git-difference-between-git-revert-checkout-and-reset/> \*\*

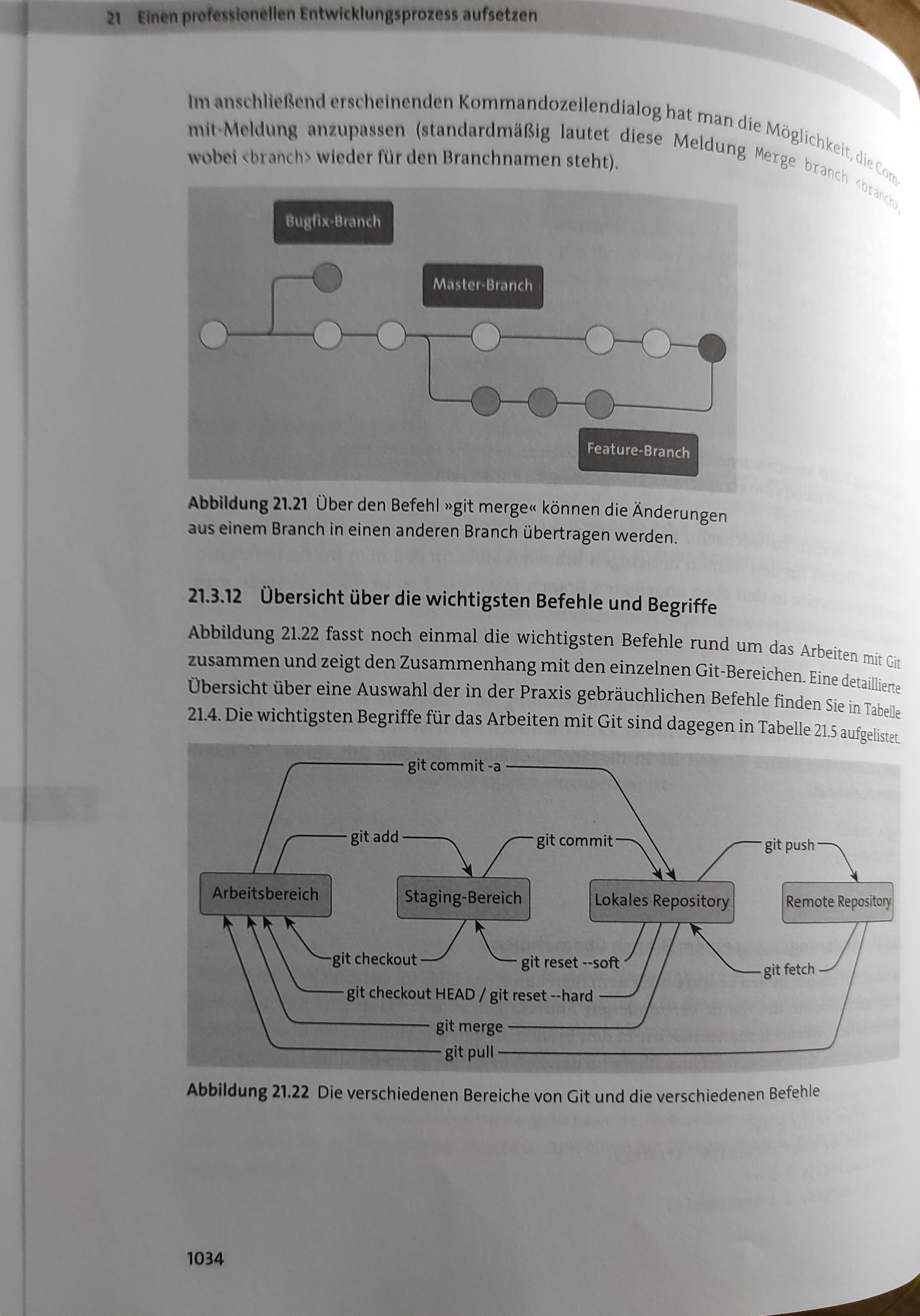
There are three commands with similar names: git reset, git restore and git revert.

* [git-revert[1]](https://git-scm.com/docs/git-revert) is about making a new commit that reverts the changes made by other commits.
* [git-restore[1]](https://git-scm.com/docs/git-restore) is about restoring files in the working tree from either the index or another commit. This command does not update your branch. The command can also be used to restore files in the index from another commit.
* [git-reset[1]](https://git-scm.com/docs/git-reset) is about updating your branch, moving the tip in order to add or remove commits from the branch. This operation changes the commit history.  
  git reset can also be used to restore the index, overlapping with git restore.

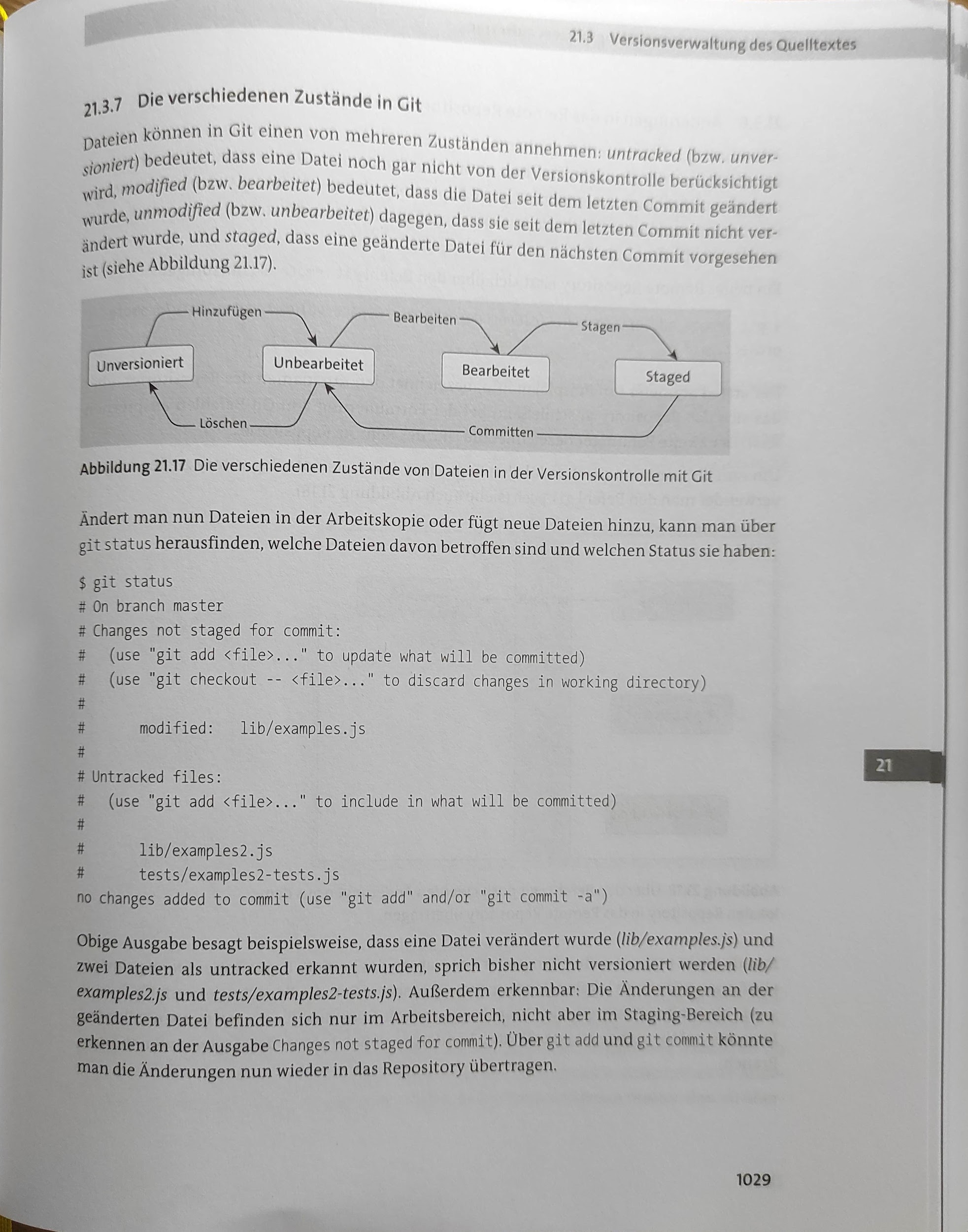
Quelle: <https://git-scm.com/docs/git> (aka **$ man git)**

<https://opensource.com/article/18/6/git-reset-revert-rebase-commands> \*\*\* Schaubilder

<https://alphanodes.com/de/git-im-praktischen-einsatz> \*\*\* Schaubild !!



Quelle: Philip Ackermann: JavaScript. Das umfassende Handbuch 3. Auflage (oder die 2.) Bereiche



Quelle: Philip Ackermann: JavaScript. Das umfassende Handbuch 3. Auflage (oder die 2.) Zustände

*21.3.12 "Die verschiedenen Bereiche von Git und die verschiedenen Befehle" \*\*\**

1. working tree
2. stage (index)
3. local repository
4. remote repository

<https://medium.com/@zjpjack/reverting-modified-in-4-stages-in-git-f3997f526902> \*\*\* ähnliches Schaubild mit Farbe und als Sequenzdiagramm

*21.3.7 "Die verschiedenen Zustände von Dateien in der Versionskontrolle mit Git" \*\*\**

1. untracked
2. unmodified
3. modified
4. staged

<https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository> \*\* noch ein Schaubild mit Farbe und als Sequenzdiagramm

TODO in einem künftigen Release dieses Dokuments sollte git tag oops in das nächste Kapitel 18 verschoben werden. Damit konzentrieren wir uns hier auf das Entfernen des commit.

### ==== githowto 18 Removing the oops tag

<https://githowto.com/remove_the_oops_tag>

<https://manpages.debian.org/bookworm/git-man/git-tag.1.en.html>

### ==== githowto 19 Changing commits

<https://githowto.com/amending_commits>

**--amend**

Replace the tip of the current branch by creating a new commit. The recorded tree is prepared as usual (including the effect of the **-i** and **-o** options and explicit pathspec), and the message from the original commit is used as the starting point, instead of an empty message, when no other message is specified from the command line via options such as **-m**, **-F**, **-c**, etc. The new commit has the same parents and author as the current one (the **--reset-author** option can countermand this).

It is a rough equivalent for:

**$ git reset --soft HEAD^**

$ ... do something else to come up with the right tree ...

**$ git commit -c ORIG\_HEAD**

but can be used to amend a merge commit.

You should understand the implications of rewriting history if you amend a commit that has already been published. (See the "RECOVERING FROM UPSTREAM REBASE" section in [git-rebase(1)](https://manpages.debian.org/bookworm/git-man/git-rebase.1.en.html).)

<https://manpages.debian.org/bookworm/git-man/git-commit.1.en.html>

**Übungsaufgabe**

Machen sie eine weitere Änderung in der Datei. Kann man den git commit --amend ein zweites Mal machen? Was passiert?

**Übungsaufgabe**

Machen sie eine weitere Änderung in der Datei. Kann man den git commit --amend ein drittes Mal machen? Was passiert?

**Übungsaufgabe**

Machen sie **KEINE** weitere Änderung in der Datei. Was passiert, wenn man git commit --amend ein weiteres Mal macht.

**Übungsaufgabe**

Machen sie eine weitere Änderung in der Datei. Aber machen sie kein git add. Was passiert, wenn man git commit --amend ein weiteres Mal macht.

**Praxistipp**

Nicht -m "MESSAGE" verwenden. Dann geht ein Editor auf und die alte MESSAGE kann ersetzt oder ergänzt werden. Alternativ kann man den git commit aus der Shell History holen.

### ==== githowto 20 Moving files

<https://githowto.com/moving_files>

<https://manpages.debian.org/bookworm/git-man/git-mv.1.en.html>

<https://manpages.debian.org/bookworm/git-man/git-rm.1.en.html>

TODO in einem künftigen Release dieses Dokuments sollte dieses Kapitel in zwei Teile aufgeteilt werden:

git mv hello.html # Teil 1

mv hello.html # Teil 2

## === Inside the git Machine: .git and git objects

### ==== githowto 21 More information about the structure

<https://githowto.com/more_structure>

### ==== githowto 22 Inside git: .git directory

<https://githowto.com/git_internals_git_directory>

[source, bash, linenums]

----

**$ tree -F .git**

.git

├── branches/

├── config

├── description

├── HEAD

├── hooks/

│ ├── applypatch-msg.sample\*

│ ├── commit-msg.sample\*

[...]

├── info/

│ └── exclude

├── objects/

│ ├── info/

│ └── pack/

└── refs/

├── heads/

└── tags/

9 directories, 17 files

----

[source, bash, linenums]

----

**$ file .git/objects/0d/576c485a252d7d544ea415cb4abcc68362bf4e**

----

Die Verzeichnisnamen entsprechen den ersten beiden Stellen der SHA-1 Hashes, welche mit **git log --oneline** angezeigt werden.

Hinweis: ls braucht die Option -C hier nicht.

Vergleiche global und local git Konfiguration:

[source, bash, linenums]

----

**$ cat ~/.gitconfig**

**$ git config --global --list**

**$ cat .git/config**

**$ git config --local --list**

**$ git config --list**

----

**Übungsaufgabe**

Legen sie im Verzeichnis **~/git\_kurs/** einen Ordner how\_to\_handle\_changes an. Dann wechseln sie in diesen Ordner. Machen sie git init.

Erstellen sie eine Datei starwars.txt mit dem Text:

Es war einmal in einer fernen Galaxy,

Fügen sie die Datei starwars.txt dem git Repository hinzu.

Nun machen sie eine Änderung in der Datei. Wie können sie die Änderung rückgängig machen?

Zeit 20 Minuten

Danach besprechen wir die Aufgabe in der Klasse. Um 14:50

TODO

Änderungen nach einem git add rückgängig machen.

Änderungen nach einem Commit rückgängig machen.

**Übungsaufgabe**

TODO

git mv

### ==== githowto 23 Git inside: Direct work with git objects

<https://githowto.com/git_internals_working_directly_with_git_objects>

<https://manpages.debian.org/bookworm/git-man/gittutorial-2.7.en.html> \*\*

-t

Instead of the content, show the object type identified by **<object>**.

-p

Pretty-print the contents of **<object>** based on its type

<https://manpages.debian.org/bookworm/git-man/git-cat-file.1.en.html>

*Git's primitives are not inherently a* [*source-code management*](https://en.wikipedia.org/wiki/Version_control) *system. Torvalds explains:*[*[62]*](https://en.wikipedia.org/wiki/Git#cite_note-63)

*In many ways you can just see git as a filesystem—it's* [*content-addressable*](https://en.wikipedia.org/wiki/Content-addressable_storage)*, and it has a notion of versioning, but I really designed it coming at the problem from the viewpoint of a filesystem person (hey, kernels is what I do), and I actually have absolutely zero interest in creating a traditional SCM system.*

*[..]*

<https://en.wikipedia.org/wiki/Git#Data_structures>

<https://youtu.be/2sjqTHE0zok> Lecture 6: Version Control (git) (2020) \*\*

plumbing versus porcelain

<https://matthew-brett.github.io/curious-git/git_object_types.html> \*\*

<https://www.geeksforgeeks.org/git-object-model/>

<https://git-scm.com/book/en/v2/Git-Internals-Git-Objects>

<https://coderefinery.github.io/git-intro/under-the-hood/>

**Übungsaufgabe**

*Try to manually find the original* **hello.html** *file from the first commit with help of SHA1 hash references in the last commit.*

<https://githowto.com/git_internals_working_directly_with_git_objects>

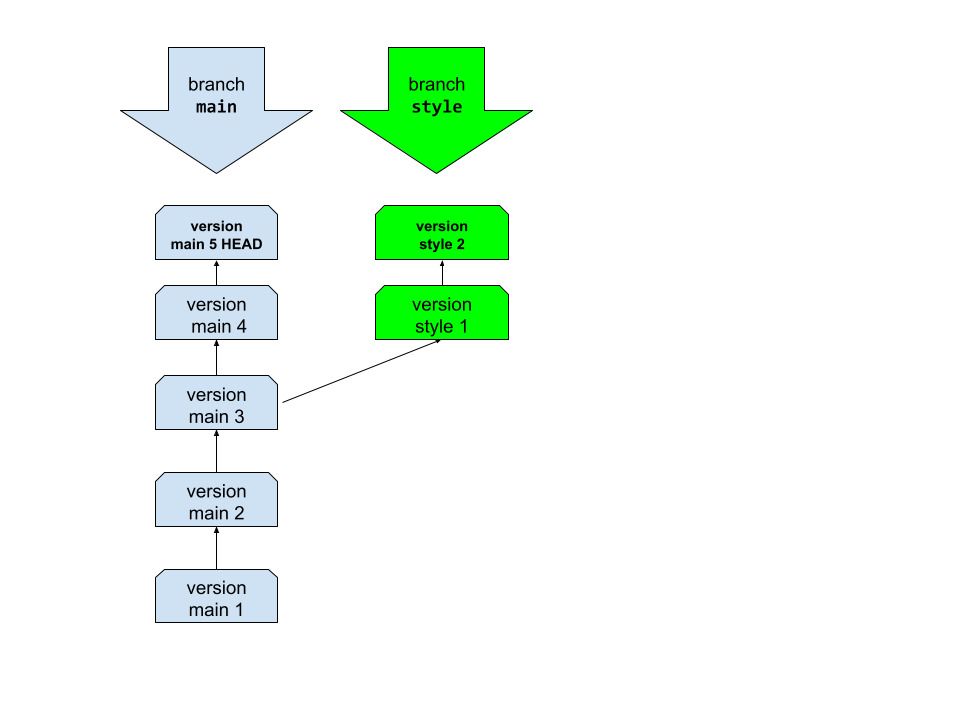
Lösung im Trainermaterial

## === Working with Branches

### ==== Ein erster Branch githowto 24 Creating a branch

<https://githowto.com/creating_a_branch>

<https://manpages.debian.org/bookworm/git-man/git-checkout.1.en.html>



Welcher branch wird gerade aktuell genutzt (und welche branches gibt es noch)?

[source, bash, linenums]

----

**$ git show-branch --list**

**$ git branch --list**

----

<https://manpages.debian.org/bookworm/git-man/git-show-branch.1.en.html>

<https://manpages.debian.org/bookworm/git-man/git-branch.1.en.html>

### ==== githowto 25 Navigating branches

<https://githowto.com/navigating_branches>

Nach dem checkout von main die Ausgabe der beiden Kommandos vergleichen:

[source, bash, linenums]

----

**$ git log --oneline**

23ca9ab (HEAD -> **style**) Updated index.html

4fe303a Hello uses style.css

00a43f5 Added css stylesheet

3e8ec5e (**main**) Added index.html

7fbef7f Moved hello.html to lib

237111d Add an author/email comment

eca36c5 (tag: v1) Added HTML header

a2e68ff (tag: v1-beta) Added standard HTML page tags

7f8f2ff Added h1 tag

856039e First Commit

----

[source, bash, linenums]

----

**$ git log --oneline --all**

23ca9ab (HEAD -> **style**) Updated index.html

4fe303a Hello uses style.css

00a43f5 Added css stylesheet

3e8ec5e (**main**) Added index.html

7fbef7f Moved hello.html to lib

237111d Add an author/email comment

eca36c5 (tag: v1) Added HTML header

a2e68ff (tag: v1-beta) Added standard HTML page tags

7f8f2ff Added h1 tag

856039e First Commit

----

Welche Objekte sind im branch main?

[source, bash, linenums]

----

**$ git ls-tree main -r**

100644 blob aeeb538cd0610f43516914c29faaae04e80a870d index.html

100644 blob 507b14220a4c57dd35c9fe8e5cf50dcc8f95e820 lib/hello.html

----

Welche Objekte sind im branch style?

[source, bash, linenums]

----

**$ git ls-tree style -r**

100644 blob 8c855bbc045a413299cda208c670a28640655dc4 index.html

100644 blob d2c586dc8e52e84693b8937c04e8e265b6a40afb lib/hello.html

100644 blob adc68fa6a4dfa0962da43093e2a0743eed439ea5 lib/style.css

----

Auch zwischen branches lassen sich Dateien vergleichen (oder mit git show anzeigen):

[source, bash, linenums]

----

**$ git diff main:lib/hello.html style:lib/hello.html**

**$ # hier sieht man aber (noch) keinen Unterschied zwischen den Dateien**

----

**Übungsaufgabe**

Aktuell haben wir branch main und style. Wechseln sie mit git checkout main in den main branch.

Legen sie einen dritten branch mit dem Namen style\_pink an und wechsel sie in diesen neuen branch (das geht mit einem einzigen Kommando). Legen sie eine Datei lib/style\_pink.css an. Mit dem Inhalt:

h1 {

color: **black**;

}

body {

background-color: **pink**;

}

Dann fügen sie das neue Stylesheet zu den Dateien index.html und hello.html hinzu.

Sie können nun alle Änderungen zusammen adden und committen.

Überprüfen sie die Änderungen mit git log --oneline --all bzw. tree -F.

Diese Aufgabe zeigt die praktische Verwendung von branches. Man kann einfach zwischen verschiedenen Stylesheets wechseln.

Zeitrahmen 20 Minuten. Nach der Frühstückspause. Das heisst ab 9:50 schauen wir uns eure Lösungen an.

### ==== githowto 26 Changes to master (now main) branch

In unserer Konfiguration ist aus master der Hauptzweig main geworden. Das haben wir ganz am Anfang in der Konfiguration so festgelegt.

<https://githowto.com/changes_in_master>

### ==== githowto 27 View the different branches

<https://githowto.com/viewing_diverging_branches>

[source, bash, linenums]

----

**$ git log --oneline --all --graph**

\* 5cd4331 (HEAD -> **main**) Added README

| \* 23ca9ab (**style**) Updated index.html

| \* 4fe303a Hello uses style.css

| \* 00a43f5 Added css stylesheet

|/

\* 3e8ec5e Added index.html

\* 7fbef7f Moved hello.html to lib

\* 237111d Add an author/email comment

\* eca36c5 (tag: v1) Added HTML header

\* a2e68ff (tag: v1-beta) Added standard HTML page tags

\* 7f8f2ff Added h1 tag

\* 856039e First Commit

----

**Übungsaufgabe: Anlegen eines neuen branch, Wechsel zwischen den branches**

Legen sie im Verzeichnis **~/git\_kurs/** einen Ordner two\_branches an. Dann wechseln sie in diesen Ordner. Machen sie git init.

A Erstellen sie eine Datei starwars\_intro.txt mit dem Text:

A long time ago in a galaxy far,

far away....

Fügen sie die Datei starwars\_intro.txt zum branch main im git Repository hinzu.

Überprüfen sie ihre Arbeit mit dem Kommando

git log --oneline --all # original command

git hist --all # alias

B Erstellen sie einen branch mit dem Namen episode1 und wechseln sie in diesen neuen branch (das geht mit einem einzigen Kommando).

Erstellen sie eine Datei episode1.txt mit dem Text:

Die Galaktische Republik wird von Unruhen erschüttert. ...

Fügen sie diese Datei dem branch episode1 hinzu.

Überprüfen sie ihre Arbeit mit dem Kommando

git log --oneline --all # original command

git hist --all # alias

C Erstellen sie eine Datei README mit dem Text:

In diesem git Repository finden sie (bald) alle Intros der StarWars Filme.

Machen sie einen checkout vom branch main. Fügen sie die Datei README dem branch main hinzu.

Überprüfen sie ihre Arbeit mit dem Kommando

git log --oneline --all --graph # original command

git hist --all --graph # alias

Zeit 30 Minuten. Bis 11:45

Danach besprechen wir die Aufgabe in der Klasse. Um 11:45

<https://en.wikipedia.org/wiki/Star_Wars_opening_crawl>

<https://www.myzitate.de/star-wars-lauftext/>

TODO

starwars\_episode2.txt

Es herrscht Aufruhr im Galaktischen Senat.

**Übungsaufgabe: einen Merge machen**

Arbeiten sie im Verzeichnis **~/git\_kurs/two\_branches/** Das ist der Ordner aus einer vorherigen Übung.

Wechseln sie in den branch main. Schauen sie sich das working directory mit tree -F an.

Dann führen sie den branch main und den branch episode1 zusammen (merge).

Kontrollieren sie das Ergebnis mit

git hist --all --graph

tree -F

Zeit 15 Minuten. Ausserdem die Pause am Nachmittag. Ab 14:20 besprechen wir die Aufgabe

## === Merging Branches

### ==== Zusammenführung von branches (ohne Konflikt) githowto 28 Merging

<https://githowto.com/merging>

<https://manpages.debian.org/bookworm/git-man/git-merge.1.en.html>

<https://www.biteinteractive.com/understanding-git-merge/> \*\*\* MUST READ

<https://manpages.debian.org/bookworm/git-man/gitworkflows.7.en.html>

<https://stackoverflow.com/questions/49715421/does-the-order-of-git-merging-matter> order of merges

<https://nvie.com/posts/a-successful-git-branching-model/> ein komplexes branching model

### ==== githowto 29 Creating a conflict

<https://githowto.com/creating_a_conflict>

**Übungsaufgabe: project "story" (merging versus rebase) status: checked**

Wir erstellen uns ein Beispiel mit drei (leeren) Dateien zum Üben von merge und rebase.

| **main** |  | **draft** |
| --- | --- | --- |
| intro |  |  |
|  |  | episode1 |
| episode2 |  |  |

[source, bash, linenums]

----

**$ mkdir ~/git\_kurs/story1**

**$ cd ~/git\_kurs/story1**

**$ git init**

**$ touch intro**

**$ git add intro**

**$ git commit -m "first commit"**

**$ git checkout -b draft**

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 11:57 intro

$ # hint: intro is now available in branch main AND draft

**$ touch episode1**

**$ git add episode1**

**$ git commit -m "added episode1"**

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:05 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 11:57 intro

**$ git checkout main**

**$ touch episode2**

**$ git add episode2**

**$ git commit -m "added episode2"**

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:07 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 11:57 intro

**$ git log --oneline --all --graph**

\* 4cc7417 (HEAD -> main) added episode2

| \* a7f7e01 (draft) added episode1

|/

\* db29c08 first commit

----

Das Beispiel ist jetzt fertig für vier Übungen. Dazu machen wir uns Kopien für jede Übung:

[source, bash, linenums]

----

**$ cd ..**

**$ cp -r story1 story2**

**$ cp -r story1 story3**

**$ cp -r story1 story4**

----

Hinweis: an Stelle der Kopien, könnten wir vor jeder neuen Übung auch einen bzw. zwei git reset --hard <sha1> machen. Oder drei mal git clone (umständlich und an dieser Stelle noch nicht erklärt).

story1 merge von draft in main <-

[source, bash, linenums]

----

**$ cd story1**

**$ git checkout main**

**$ ls -l**

$ ..

**$ git log --oneline --all --graph**

\* 4cc7417 (HEAD -> main) added episode2

| \* a7f7e01 (draft) added episode1

|/

\* db29c08 first commit

**$ git merge draft**

**$ git log --oneline --all --graph**

\* d269cc7 (HEAD -> main) Merge branch 'draft'

|\

| \* a7f7e01 (draft) added episode1

\* | 4cc7417 added episode2

|/

\* db29c08 first commit

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:17 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 12:07 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 11:57 intro

----

Ergebnis: die Datei episode1 ist jetzt auch in branch main.

Im git log ist der Änderungsverlauf der branches (der merge) sichtbar.

Wie sieht der Inhalt von branch draft aus?

[source, bash, linenums]

----

**$ git checkout draft**

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:17 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 11:57 intro

----

Hinweis: der branch in dem man sich befindet wird auch **your side** genannt. Der branch welcher gemerged werden soll wird auch **their side** genannt.

story2 merge von main in draft ->

[source, bash, linenums]

----

**$ cd ..**

**$ cd story2**

**$ git checkout draft**

**$ ls -l**

$ ..

**$ git log --oneline --all --graph**

\* 4cc7417 (main) added episode2

| \* a7f7e01 (HEAD -> draft) added episode1

|/

\* db29c08 first commit

**$ git merge main**

**$ git log --oneline --all --graph**

\* d9b3be6 (HEAD -> draft) Merge branch 'main' into draft

|\

| \* 4cc7417 (main) added episode2

\* | a7f7e01 added episode1

|/

\* db29c08 first commit

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:39 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 12:42 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 12:11 intro

----

Ergebnis: die Datei episode2 ist jetzt auch in branch draft.

Im git log ist der Änderungsverlauf der branches (der merge) sichtbar.

Wie sieht der Inhalt von branch main aus?

[source, bash, linenums]

----

**$ git checkout main**

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:42 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 12:11 intro

----

story3 rebase von draft in main <-

[source, bash, linenums]

----

**$ cd ..**

**$ cd story3**

**$ git checkout main**

**$ ls -l**

$ ..

**$ git log --oneline --all --graph**

\* 4cc7417 (HEAD -> main) added episode2

| \* a7f7e01 (draft) added episode1

|/

\* db29c08 first commit

**$ git rebase draft**

**$ git log --oneline --all --graph**

\* 4f0e5cd (HEAD -> main) added episode2

\* a7f7e01 (draft) added episode1

\* db29c08 first commit

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:52 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 12:52 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 12:11 intro

----

Ergebnis: die Datei episode1 ist jetzt auch in branch main.

Im git log ist der Änderungsverlauf der branches (der merge) **nicht** sichtbar.

story4 rebase von main in draft ->

[source, bash, linenums]

----

**$ cd ..**

**$ cd story4**

**$ git checkout draft**

**$ ls -l**

$ ..

**$ git log --oneline --all --graph**

\* 4cc7417 (main) added episode2

| \* a7f7e01 (HEAD -> draft) added episode1

|/

\* db29c08 first commit

**$ git rebase main**

**$ git log --oneline --all --graph**

\* 68cb6aa (HEAD -> draft) added episode1

\* 4cc7417 (main) added episode2

\* db29c08 first commit

**$ ls -l**

total 0

-rw-r--r-- 1 alice alice 0 Sep 9 12:55 episode1

-rw-r--r-- 1 alice alice 0 Sep 9 12:55 episode2

-rw-r--r-- 1 alice alice 0 Sep 9 12:11 intro

----

Ergebnis: die Datei episode2 ist jetzt auch in branch draft.

Im git log ist der Änderungsverlauf der branches (der merge) **nicht** sichtbar.

**Abschlusstest story4**

Der Inhalt von branch main soll genauso aussehen, wie der Inhalt von branch draft. Was tun?

Lösung im Trainermaterial

**Übung einen Merge Conflict herstellen und bearbeiten (imagine)**

Imagine there's no heaven

It's easy if you try

No hell below us

Above us only sky

<https://de.wikipedia.org/wiki/Imagine_(Lied)>

[source, bash, linenums]

----

**$ mkdir ~/git\_kurs/merge\_conflict\_imagine**

**$ cd ~/git\_kurs/merge\_conflict\_imagine**

**$ git init**

**$ nano imagine.txt**

Imagine there's no heaven

**$ git add imagine.txt**

**$ git commit -m "version 0"**

**$ git checkout -b draft**

**$ nano imagine.txt**

It's easy if you try

No hell below us

**$ git add imagine.txt**

**$ git commit -m "version 1"**

**$ git checkout main**

**$ nano imagine.txt**

Above us only sky

**$ git add imagine.txt**

**$ git commit -m "version 2"**

**$ git log --oneline --all --graph**

\* 7fe4c72 (HEAD -> main) version 2

| \* ba2d02e (draft) version 1

|/

\* 634c821 version 0

**$ git merge draft**

Auto-merging imagine.txt

CONFLICT (content): Merge conflict in imagine.txt

Automatic merge failed; fix conflicts and then commit the result.

----

[source, bash, linenums]

----

**$ cat imagine.txt**

Imagine there's no heaven

<<<<<<< HEAD

Above us only sky

=======

It's easy if you try

No hell below us

>>>>>>> draft

----

[source, bash, linenums]

----

**$ nano imagine.txt**

Imagine there's no heaven

It's easy if you try

No hell below us

Above us only sky

**$ git add imagine.txt**

**$ git commit -m "version 3 merged draft into main"**

[main 778eda2] version 3 merged draft into main

**$ git log --oneline --all --graph**

\* 778eda2 (HEAD -> main) version 3 merged draft into main

|\

| \* ba2d02e (draft) version 1

\* | 7fe4c72 version 2

|/

\* 634c821 version 0

----

Falls notwendig, kann man nun auch in den branch draft wechseln und main dazu mergen. Es gibt dabei keinen Konflikt:

[source, bash, linenums]

----

**$ git checkout draft**

**$ git log --oneline --all --graph # TODO**

**$ git merge main**

Updating ba2d02e..778eda2

Fast-forward

imagine.txt | 1 +

1 file changed, 1 insertion(+)

**$ git log --oneline --all --graph**

\* 778eda2 (HEAD -> draft, main) version 3 merged draft into main

|\

| \* ba2d02e version 1

\* | 7fe4c72 version 2

|/

\* 634c821 version 0

----

<https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts#:~:text=Git%20can%20handle%20most%20merges,working%20in%20a%20team%20environment>.

<https://www.atlassian.com/git/tutorials/using-branches/merge-strategy>

<https://stackoverflow.com/questions/44359334/what-are-the-reasons-and-cases-that-cause-git-merge-conflicts> \*\*\* must read before going on with creating merge conflicts.

merging theory: diff3 three way merge \*\*\*

<https://en.wikipedia.org/wiki/Merge_(version_control)>

<https://en.wikipedia.org/wiki/Branching_(version_control)>

<https://en.wikipedia.org/wiki/Diff3>

<https://www.gnu.org/software/diffutils/manual/html_node/diff3-Merging.html>

<https://www.howtoforge.com/tutorial/how-to-compare-three-files-in-linux-using-diff3-tool/> --

<https://www.geeksforgeeks.org/diff3-command-in-linux-with-examples/> --

**Übungsaufgabe: einen Merge Conflict bearbeiten (starwars\_intro) TODO**

Wechseln sie in den branch episode1 und ändern sie in der Datei starwars\_intro.txt die erste Zeile. Sie können sie zum Beispiel ins deutsche übersetzen. Dann adden und committen. TODO das müssen wir ändern, siehe übernächster Absatz.

Wechseln sie zurück in den branch main, machen sie einen merge. Das sollte einen merge conflict geben. Beheben sie den Konflikt. Dann checken sie die Änderungen ins git repository.

Hinweis: falls es keinen merge conflict geben sollte, überlegen sie bitte, wie kann man einen merge conflict erzeugen. Das gibt keinen Konflikt. Wir brauchen eine Änderung der Datei starwars\_intro.txt im branch main und eine Änderung der Datei starwars\_intro.txt im branch episode1. Die Änderungen müssen in der gleichen Zeile sein. Wir könnten z.B. jeweils eine dritte Zeile hinzufügen, einmal in Englisch und einmal in Deutsch.

20 Minuten. Wir besprechen das um 14:55

<https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts> \*\*

### ==== githowto 30 Resolving conflicts

<https://githowto.com/resolving_conflicts>

Graphical merge tools **apt install meld diffuse**

**vimdiff**

How to cancel a merge operation:

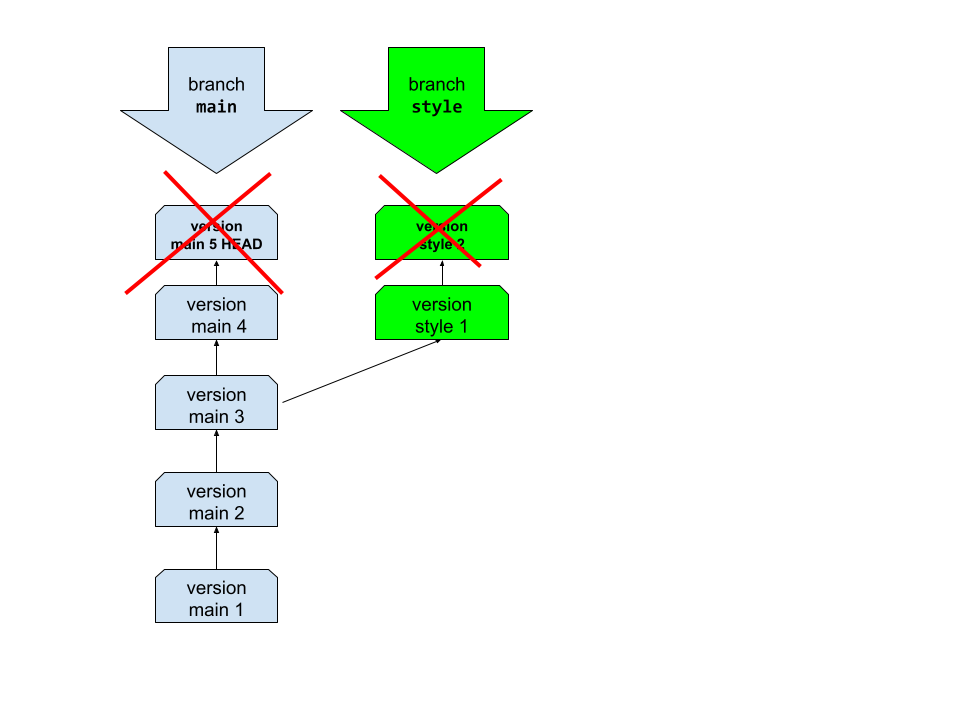
<https://www.freecodecamp.org/news/git-abort-merge-how-to-cancel-a-merge-in-git/>

## === Rebasing Branches (Preparation)

### ==== githowto 31 Relocating as an alternative to merging

<https://githowto.com/rebasing_vs_merging>

Rewriting history. Vorbereitungsarbeiten vor dem relocating (rebase):



[source, bash, linenums]

----

**$ git hist**

\* 645c4e6 2011-03-09 | Merged main fixed conflict. (HEAD, style) [Alexander Shvets]

|\

| \* 454ec68 2011-03-09 | Life is great! (main) [Alexander Shvets]

\* | 5813a3f 2011-03-09 | Merge branch 'main' into style [Alexander Shvets]

|\ \

| |/

| \* 6c0f848 2011-03-09 | Added README [Alexander Shvets]

\* | 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]

\* | 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

\* | 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

|/

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

----

Quelle: <https://githowto.com/rebasing_vs_merging> modified by WH (master -> main)

Hier ist farblich markiert, ab wo die Zweige in main und style resetted (entfernt) werden. Das passiert ind zwei getrennten Schritten. In den Kapiteln 32 (reset style) und 33 (reset main). Mit dem eigentlichen Thema rebase geht es dann in Kapitel 34 los.

### ==== githowto 32 Resetting the style branch

<https://githowto.com/resetting_the_greet_branch>

### ==== githowto 33 Reset of the master [now main] branch

<https://githowto.com/resetting_the_master_branch>

### === Rebasing Branches

### ==== githowto 34 Rebase

<https://githowto.com/rebasing>

<https://manpages.debian.org/bookworm/git-man/git-rebase.1.en.html>

<https://www.atlassian.com/git/tutorials/merging-vs-rebasing> \*\*\*

<https://zwischenzugs.com/2016/04/24/interactive-git-tutorials-rebase-and-bisect/>

<https://timmousk.com/blog/git-rebase-abort/>

<https://www.warp.dev/terminus/undo-a-git-rebase>

### ==== githowto 35 Merging to the master (now main) branch

<https://githowto.com/merging_back_to_master>

Als erstes nochmal **git log --oneline --graph --all** damit man den Unterschied nach dem merge sieht.

### === Teamwork with (Remote) Repositories: clone, push, pull

### ==== githowto 36 Multiple Repositories

<https://githowto.com/mutliple_repositories>

### ==== githowto 37 Cloning repositories

<https://githowto.com/cloningrepositories>

<https://manpages.debian.org/bookworm/git-man/git-clone.1.en.html>

<https://www.atlassian.com/de/git/tutorials/setting-up-a-repository/git-clone> \*\* Schaubild

<https://devconnected.com/how-to-clone-a-git-repository/>

### ==== githowto 38 Examine the cloned history

<https://githowto.com/review_the_cloned_repository>

Wie sehen die Unterschiede zum Original (origin) aus?

[source, bash, linenums]

----

**$ git status**

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

----

[source, bash, linenums]

----

**$ git show-branch --list**

\* [main] Added shared comment to readme

**$ git show-branch --list --all**

\* [main] Added shared comment to readme

[origin/HEAD] Added shared comment to readme

[origin/main] Added shared comment to readme

[origin/style] Updated index.html

----

TODO above output is not the original from githowto.

Der Verweis auf die Quelle dieses Repositories (aka origin) findet sich in der Datei **.git/config** .

[source, bash, linenums]

----

**$ cat .git/config**

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

[remote "origin"]

url = [...]/git\_tutorial/work/hello

fetch = +refs/heads/\*:refs/remotes/origin/\*

[branch "main"]

remote = origin

merge = refs/heads/main

----

[source, bash, linenums]

----

**$ git branch -a**

\* main

remotes/origin/HEAD -> origin/main

remotes/origin/main

remotes/origin/style

----

In der Datei **.git/config** im originalen Verzeichnis hello, findet sich kein Verweis auf das geclonte Repository. Ausserdem:

[source, bash, linenums]

----

**$ git branch -a**

\* main

style

----

### ==== githowto 39 What is origin?

<https://githowto.com/what_is_origin>

<https://manpages.debian.org/bookworm/git-man/git-remote.1.en.html>

Siehe auch das Schaubild hier:

<https://www.baeldung.com/git-guide> 7. Git Workflow

<https://www.atlassian.com/git/tutorials/syncing> \*\*\*

[source, bash, linenums]

----

**$ git remote --verbose**

origin [...]/git\_kurs/git\_tutorial/work/hello (fetch)

origin [...]/git\_kurs/git\_tutorial/work/hello (push)

----

Hinweis: wie sehen **git remote** und **git branch -a** unter origin aus?

### ==== githowto 40 Remote branches

<https://githowto.com/remote_branches>

<https://manpages.debian.org/bookworm/git-man/git-branch.1.en.html>

Praxistipp

Detailierte Informationen zum remote branch anzeigen lassen --verbose bzw. -v :

[source, bash, linenums]

----

**$ git remote -verbose** # output modified WH

origin [..]/git\_kurs/git\_tutorial/work/hello (fetch)

origin [..]/git\_kurs/git\_tutorial/work/hello (push)

shared ../hello.git (fetch)

shared ../hello.git (push)

----

Detailierte Informationen zum remote branch anzeigen lassen --all :

[source, bash, linenums]

----

**$ git branch --list --all --verbose**

\* main dca6037 Added shared comment to readme

style 64327db Updated index.html

remotes/shared/main dca6037 Added shared comment to readme

----

<https://www.hostinger.com/tutorials/how-to-rename-a-git-branch/>

Wie können wir uns aus dem clone das Original (origin) anzeigen lassen?

[source, bash, linenums]

----

**$ git show origin/main:README**

This is the Hello World example from the git tutorial.

**$ git show origin/style:lib/style.css**

h1 {

color: red;

}

----

### ==== githowto 41 Changing the original repository

<https://githowto.com/change_the_original_repository>

Im geklonten repository:

[source, bash, linenums]

----

**$ git diff origin/HEAD:README HEAD:README**

diff --git a/README b/README

index d2d45ac..74aa19c 100644

--- a/README

+++ b/README

@@ -1,2 +1,3 @@

This is the Hello World example from the git tutorial.

(changed in original)

+(Changed in the original and pushed to shared)

----

### ==== githowto 42 Fetching changes

<https://githowto.com/fetching_changes>

<https://manpages.debian.org/bookworm/git-man/git-fetch.1.en.html>

### ==== githowto 43 Merging pulled changes

<https://githowto.com/merging_pulled_changes>

Warum in zwei Schritten ( fetch und merge). Damit man erstmal die anstehenden Änderungen anschauen kann.

### ==== githowto 44 Pulling and merging changes

<https://githowto.com/pulling_changes>

<https://manpages.debian.org/bookworm/git-man/git-pull.1.en.html>

Hinweis: man könnte hier das githowto Beispiel komplett für ein git pull erweitern.

### ==== githowto 45 Adding a tracking branch

<https://githowto.com/adding_a_tracking_branch>

[source, bash, linenums]

----

**$ git show-branch --list**

\* [main] Changed README in original repo

[style] Updated index.html

----

Hätte man gleich beim clone alle branches angeben können?

**Übungsaufgabe**

Wir haben heute im Verzeichnis **~/git\_kurs/story1** gearbeitet. Wechseln sie in das Verzeichnis. Dann ein Verzeichnis nach oben. Machen sie einen Clone von story1. Der Clone soll heissen **new\_story1**. Arbeiten sie die Kapitel

36 bis 45 aus dem githowto <https://githowto.com/mutliple_repositories> mit selbstgewählten Beispielen durch. Machen sie einen fetch und merge. Ausserdem einen pull (aber die Änderungen im Original Branch draft machen).

Wichtiger Hinweis: bitte achten sie unbedingt bei jeder einzelnen Aktion darauf, wo sie arbeiten. In welchem **Verzeichnis** arbeiten sie? In welchem **repository** arbeiten sie (original oder clone) Welchen **branch** im git repository bearbeiten sie? Wo ist der aktuelle **HEAD**? Sind alle **Änderungen** ge-added und committed?

Zeitrahmen 40 Minuten. Wir besprechen diese Aufgabe um 14:50.

**Übungsaufgabe**

**$ git clone alice@<ADDRESS>:~/git\_kurs/git\_tutorial/work/hello ~/Downloads/cloned\_hello\_wh**

### === Bare Repos ("central" git server)

### ==== githowto 46 Bare repos

<https://githowto.com/bare_repositories>

Das bare repository kann nicht als work tree genutzt werden:

[source, bash, linenums]

----

**$ git status**

fatal: this operation must be run in a work tree

**$ touch test**

**$ git add test**

fatal: this operation must be run in a work tree

----

[source, bash, linenums]

----

**$ git log --oneline --all --graph**

\* bfb7f16 (HEAD -> main) Changed README in original repo

\* 64327db (style) Updated index.html

\* 0f648fe Hello uses style.css

\* a27c627 Added css stylesheet

\* 5cd4331 Added README

\* 3e8ec5e Added index.html

\* 7fbef7f Moved hello.html to lib

\* 237111d Add an author/email comment

\* eca36c5 (tag: v1) Added HTML header

\* a2e68ff (tag: v1-beta) Added standard HTML page tags

\* 7f8f2ff Added h1 tag

\* 856039e First Commit

----

[source, bash, linenums]

----

**$ git show-branch --list**

\* [main] Changed README in original repo

[style] Updated index.html

----

Wozu braucht man ein bare repo? Kein working tree, kein .git Verzeichnis.

Zum besseren Verständnis machen wir vor der Übung aus dem githowto ein clone von einem Projekt bei github. Wir nehmen oh-my-zsh.

<https://github.com/ohmyzsh/ohmyzsh>

Hinweis: Wir schauen uns das git log an.

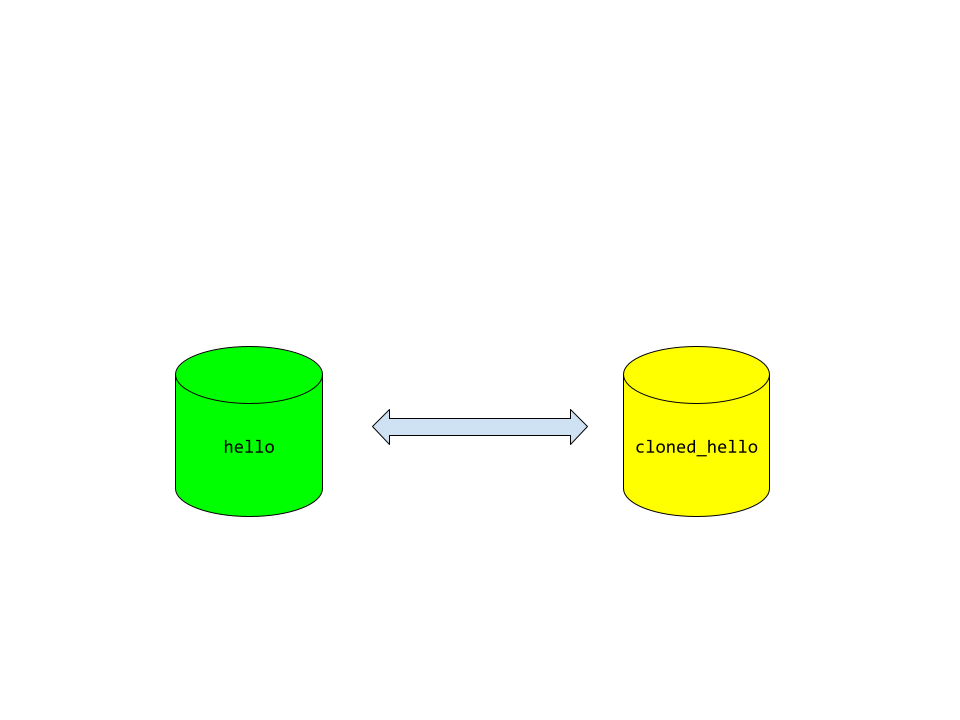
<https://www.saintsjd.com/2011/01/what-is-a-bare-git-repository/> \*\*

<https://www.atlassian.com/de/git/tutorials/setting-up-a-repository/git-clone> \*\* Schaubild!

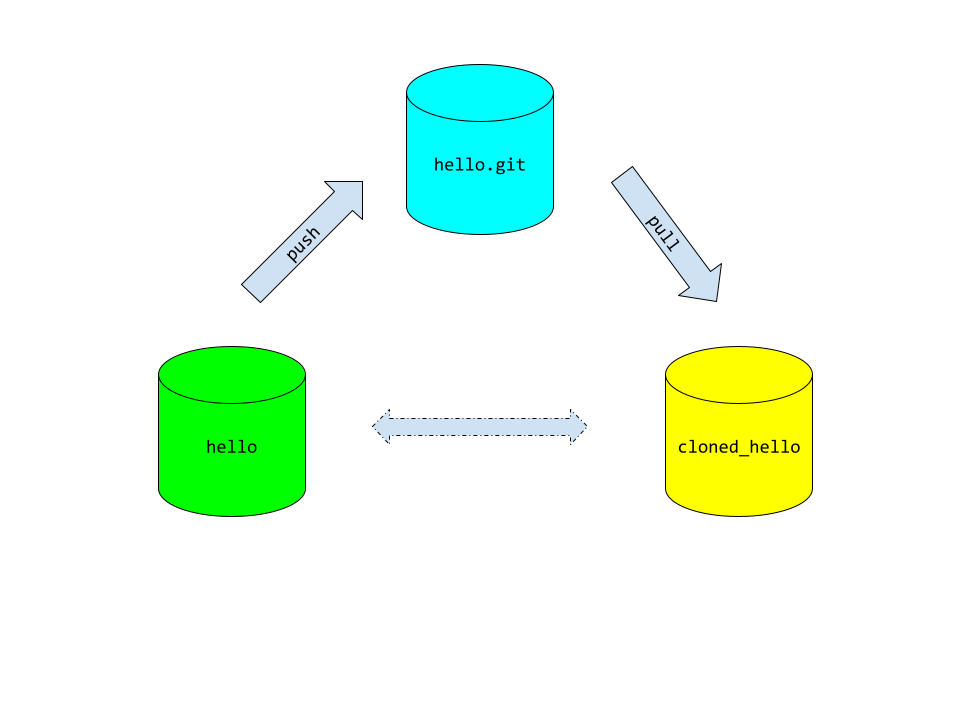
Vergleiche **git init --bare**. Kann man das als Übungsaufgabe machen?

Praxistipp

<https://www.atlassian.com/git/tutorials/dotfiles>



===



### ==== githowto 47 Adding a remote repository

<https://githowto.com/adding_a_remote_repository>

<https://manpages.debian.org/bookworm/git-man/git-remote.1.en.html>

Warum machen wir das? Verbindung zum "git server". Der Verbindung geben wir den Namen "shared".

[source, bash, linenums]

----

**$ cat .git/config**

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

[remote "shared"]

url = ../hello.git

fetch = +refs/heads/\*:refs/remotes/shared/\*

----

[source, bash, linenums]

----

**$ git remote show shared**

\* remote shared

Fetch URL: ../hello.git

Push URL: ../hello.git

HEAD branch: main

Remote branches:

main new (next fetch will store in remotes/shared)

style new (next fetch will store in remotes/shared)

Local refs configured for 'git push':

main pushes to main (up to date)

style pushes to style (up to date)

----

Im hello.git:

[source, bash, linenums]

----

**$ cat config**

[core]

repositoryformatversion = 0

filemode = true

bare = true

[remote "origin"]

url = [..]/git\_tutorial/work/hello

----

**Übungsaufgabe**

Wechseln sie mit dem cd Befehl in das Verzeichnis **hello.git**. Lassen sie sich mit **ls -l** den Inhalt des Verzeichnisses anzeigen. Wir haben gerade die geänderte Datei **README** in dieses Repository gepusht. Warum ist sie im work tree nicht zu sehen? Wer schafft es, sich den Inhalt der Datei **README** anzeigen zu lassen?

### ==== githowto 48 Submitting changes

<https://githowto.com/pushing_a_change>

<https://manpages.debian.org/bookworm/git-man/git-push.1.en.html>

Wie sieht **../hello.git** aus? Immer noch kein working tree und kein .git Verzeichnis! Gut so!

### ==== githowto 49 ~~Removing~~ [extract] common changes

<https://githowto.com/pulling_shared_changes>

<https://manpages.debian.org/bookworm/git-man/git-pull.1.en.html>

<https://www.warp.dev/terminus/git-undo-pull> \*\*\*

**Übungsaufgabe**

Erstellen sie im **hello** Repository im branch **main** eine Datei **impressum.html** mit diesem Inhalt:

Inhaber: Max Mustermann

Fügen sie die Datei dem Repository hinzu (add, commit).

Pushen sie die Änderungen zu **hello.git**

Holen sie die Änderungen in das **cloned\_hello** Repository.

Überprüfen sie, den Inhalt der Datei impressum.html im cloned\_hello Repository. Lassen sie sich auch das git log anzeigen.

Sie können die entsprechenden Befehle in den drei Kapiteln 47 - 49 aus dem githowto nachsehen. Achtung: es werden nicht mehr alle Befehle gebraucht. Die Repositories sind ja schon miteinander verknüpft.

Zeitrahmen 30 Minuten. Bis 10:25.

TODO Änderungen in cloned\_hello nach hello übertragen. Funktioniert!

**Übungsaufgabe: arbeiten mit einem zentralen git repository im Klassenraum**

Vorbereitung **Trainer**

Ein Verzeichnis **~/git\_kurs/masterplan.git** anlegen. In das Verzeichnis wechseln und ein git bare repository anlegen.

[source, bash, linenums]

----

**$ mkdir ~/git\_kurs/masterplan.git**

**$ cd ~/git\_kurs/masterplan.git**

**$ git init --bare**

----

Aufgabe **Teilnehmer**

Holen sie sich einen clone des o.g. Repositories. Da sind noch keine Projekdaten drin, aber das werden wir ändern.

[source, bash, linenums]

----

**$ cd ~/git\_kurs/**

**$ git clone alice@werner:~/git\_kurs/masterplan.git**

**$ cd masterplan**

**$ ls -lA**

total 4

drwxr-xr-x 7 wehe wehe 4096 Sep 11 18:01 .git

----

Legen sie eine Datei an, fügen sie diese Datei ihrem lokalen Repository hinzu und übertragen sie die Datei zu masterplan.git.

Holen sie sich den aktuellen Stand von masterplan.git

$ git config pull.rebase true

Erst pull dann push.

Praxisrelevanz des Beispiels: Nicht jede Firma möchte ihre git repositories auf einem öffentlichen Server (github, gitlab, …) lagern.

### ==== githowto 50 Placing your git repository [git server (daemon)]

<https://githowto.com/hosting_your_git_repositories>

<https://manpages.debian.org/bookworm/git-man/git-daemon.1.en.html>

Hinweis: githowto 50, 51 können wir im Kurs entfallen lassen. Weil wir stattdessen github als Server nehmen.

Welche Portnummern, Dateien, etc. benutzt der git-daemon (output modified WH)?

[source, bash, linenums]

----

**$ lsof -c git-daemon -P**

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

git-daemo 224060 wehe cwd DIR 8,8 4096 14786710 [..]/git\_kurs/git\_tutorial/work

git-daemo 224060 wehe rtd DIR 8,7 4096 2 /

git-daemo 224060 wehe txt REG 8,7 2129696 1322890 /usr/lib/git-core/git-daemon

git-daemo 224060 wehe mem REG 8,7 3407104 1312880 /usr/lib/locale/locale-archive

git-daemo 224060 wehe mem REG 8,7 1922136 1320789 /usr/lib/x86\_64-linux-gnu/libc.so.6

git-daemo 224060 wehe mem REG 8,7 121280 1316781 /usr/lib/x86\_64-linux-gnu/libz.so.1.2.13

git-daemo 224060 wehe mem REG 8,7 629384 1313684 /usr/lib/x86\_64-linux-gnu/libpcre2-8.so.0.11.2

git-daemo 224060 wehe mem REG 8,7 210968 1310177 /usr/lib/x86\_64-linux-gnu/ld-linux-x86-64.so.2

git-daemo 224060 wehe 0u CHR 136,1 0t0 4 /dev/pts/1

git-daemo 224060 wehe 1u CHR 136,1 0t0 4 /dev/pts/1

git-daemo 224060 wehe 2u CHR 136,1 0t0 4 /dev/pts/1

git-daemo 224060 wehe 3u IPv4 1025537 0t0 TCP \*:9418 (LISTEN)

git-daemo 224060 wehe 4u IPv6 1025538 0t0 TCP \*:9418 (LISTEN)

----

Bildschirmausgabe des git-daemon:

[source, bash, linenums]

----

**$ git daemon --verbose --export-all --base-path=.**

[224060] Ready to rumble

[224638] Connection from [::1]:34878

[224638] Extended attribute "host": localhost

[224638] Extended attribute "protocol": version=2

[224638] Request upload-pack for '/hello.git'

[224060] [224638] Disconnected

[226788] Connection from 192.168.22.110:34364

[226788] Extended attribute "host": 192.168.22.110

[226788] Extended attribute "protocol": version=2

[226788] Request upload-pack for '/hello.git'

[224060] [226788] Disconnected

…

----

### ==== githowto 51 Sharing repositories

<https://githowto.com/sharing_repos>

**Übungsaufgabe**

Statt localhost aus githowto 50 geht auch die externe IPv4 oder IPv6 Adresse   
**ip -brief address show**. Das machen wir als Übung.

**Übungsaufgabe**

*If you want to allow push to the repository Git Daemon, add* **--enable=receive-pack** *tag to* **git daemon** *command. Be attentive, this server does not perform authentication, so anyone can push changes to your repository.*

<https://githowto.com/hosting_your_git_repositories>

**$ git daemon --verbose --export-all --base-path=. --enable=receive-pack**

Alle git Server im Klassenraum finden:

# nmap -p 9418 10.0.9.0/24

SSH funktioniert hier mit Passwort. Das funktioniert ohne git daemon. Ist aber umständlicher zu schreiben, hat dafür Authentifizierung.

**$ git clone alice@localhost:~/git\_kurs/git\_tutorial/work/hello.git network\_hello**

$ git clone <user>@<hostname>:<account|path>/<reponame>.git

<https://www.atlassian.com/de/git/tutorials/setting-up-a-repository/git-clone> \*\*

### ==== githowto Thank you!

<https://githowto.com/thank_you>

## === Beyond the githowto:

### ==== git fsck

<https://manpages.debian.org/bookworm/git-man/git-fsck.1.en.html>

[source, bash, linenums]

----

**$ git fsck**

Checking object directories: 100% (256/256), done.

dangling tree 5a66444a49c945405061dd9af09bcd575cadd392

----

### ==== git gc - garbage collection

[source, bash, linenums]

----

**$ git gc**

Enumerating objects: 62, done.

Counting objects: 100% (62/62), done.

Delta compression using up to 4 threads

Compressing objects: 100% (50/50), done.

Writing objects: 100% (62/62), done.

Total 62 (delta 12), reused 0 (delta 0), pack-reused 0

----

<https://manpages.debian.org/bookworm/git-man/git-gc.1.en.html>

<https://github.blog/2022-09-13-scaling-gits-garbage-collection/>

<https://www.geeksforgeeks.org/git-gc-garbage-collection/>

<https://www.atlassian.com/git/tutorials/git-gc>

### ==== submodules

[source, bash, linenums]

----

*A submodule is a repository embedded inside another repository. The submodule has its own history; the repository it is embedded in is called a superproject.*

*On the filesystem, a submodule usually (but not always - see FORMS below) consists of (i) a Git directory located under the* ***$GIT\_DIR/modules/*** *directory of its superproject, (ii) a working directory inside the superproject’s working directory, and a* ***.git*** *file at the root of the submodule’s working directory pointing to (i).*

----

<https://manpages.debian.org/bookworm/git-man/gitsubmodules.7.en.html>

<https://manpages.debian.org/bookworm/git-man/git-submodule.1.en.html>

*Git submodules allow you to keep a git repository as a subdirectory of another git repository. Git submodules are simply a reference to another repository at a particular snapshot in time. Git submodules enable a Git repository to incorporate and track version history of external code.*

<https://www.atlassian.com/git/tutorials/git-submodule> \*\*

Aufbau der .gitmodules Datei

[source, bash, linenums]

~.gitmodules

----

[submodule "hello"]

path = hello

url = wehe@localhost:~/git\_kurs/git\_tutorial/work/hello

----

<https://manpages.debian.org/bookworm/git-man/gitmodules.5.en.html>

caveats e.g.

*If you want to clone a repository including its submodules you can use the* **--recursive** *parameter.*

<https://www.vogella.com/tutorials/GitSubmodules/article.html> \*\*

<https://matthew-brett.github.io/curious-git/git_submodules.html> \*\*

### ==== .gitignore

[source, bash, linenums]

----

*A* ***gitignore*** *file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected; see the NOTES below for details.*

*Each line in a* ***gitignore*** *file specifies a pattern. When deciding whether to ignore a path, Git normally checks* ***gitignore*** *patterns from multiple sources, with the following order of precedence, from highest to lowest (within one level of precedence, the last matching pattern decides the outcome): [..]*

----

<https://manpages.debian.org/bookworm/git-man/gitignore.5.en.html>

### ==== git stash

<https://manpages.debian.org/bookworm/git-man/git-stash.1.en.html>

### ==== git worktree

<https://opensource.com/article/21/4/git-worktree>

### ==== undo

<https://zwischenzugs.com/2020/09/10/five-ways-to-undo-a-commit-in-git/> \*\*

<https://www.atlassian.com/git/tutorials/undoing-changes> \*\*\*

Kofler Kap. 3.4

### ==== FAQ, Tipps and Tricks

**Show all files in current branch, starting recursively from HEAD.**

[source, bash, linenums]

----

**$ git ls-tree --full-tree -r HEAD**

100644 blob 74aa19c8d3ee054b6d11c33172c96ddda27a8821 README

100644 blob 8c855bbc045a413299cda208c670a28640655dc4 index.html

100644 blob d2c586dc8e52e84693b8937c04e8e265b6a40afb lib/hello.html

100644 blob adc68fa6a4dfa0962da43093e2a0743eed439ea5 lib/style.css

----

**More searching tools:**

<https://git-scm.com/book/en/v2/Git-Tools-Searching>

<https://opensource.com/article/21/4/git-whatchanged> \*\*

**new commands**

checkout versus switch (new)

reset … versus restore (new)

<https://www.freecodecamp.org/news/7-git-commands-you-might-not-know/>

**Where am I?**

[source, bash, linenums]

----

**$ cd lib**

**$ git rev-parse --is-inside-work-tree**

----

[source, bash, linenums]

----

**$ cd .git**

**$ git rev-parse --is-inside-git-dir**

----

Bereinigung der History

<https://www.geeksforgeeks.org/git-squash/>

**extras**

----

**# apt install git-extras**

----

**cherry-pick**

<https://www.atlassian.com/de/git/tutorials/cherry-pick>

<https://git-scm.com/docs/git-cherry-pick>

**copy branch, test something, delete branch**

$ git branch -c test\_branch

$ # do some tests

$ git branch -d test\_branch

**unsorted**

gitinspector.deb

<https://zaiste.net/posts/how_i_use_git_-_basics/>

<https://zaiste.net/posts/nifty_ssh_trick_named_servers/>

<https://stackoverflow.com/questions/1964142/how-can-i-list-all-the-different-versions-of-a-file-and-diff-them-also>

<https://daniel.haxx.se/blog/2020/11/09/this-is-how-i-git/>

<https://xkcd.com/1597/>

<https://www.heise.de/hintergrund/Die-vielfaeltigen-Faehigkeiten-von-Git-Teil-1-4456122.html?seite=all> \*\*\*

<https://www.heise.de/hintergrund/Die-vielfaeltigen-Funktionen-von-Git-Teil-2-4476370.html?seite=all> \*\*\*

<https://ohshitgit.com/de>

### ==== Glossar

*Git does not impose many restrictions on how it should be used, but some conventions are adopted in order to organize histories, especially those which require the cooperation of many contributors.*

* *The* ***master*** *branch is created by default with git init* [*[107]*](https://en.wikipedia.org/wiki/Git#cite_note-109) *and is often used as the branch that other changes are merged into.*[*[108]*](https://en.wikipedia.org/wiki/Git#cite_note-110) *Correspondingly, the default name of the upstream remote is* ***origin*** *and so the name of the default remote branch is* ***origin/master****. The use of master as the default branch name is not universally true. Repositories created in GitHub and GitLab will initialize with a* ***main*** *branch instead of master.* [*[109]*](https://en.wikipedia.org/wiki/Git#cite_note-111)[*[110]*](https://en.wikipedia.org/wiki/Git#cite_note-112)
* *Pushed commits should usually not be overwritten, but should rather be reverted*[*[111]*](https://en.wikipedia.org/wiki/Git#cite_note-113) *(a commit is made on top which reverses the changes to an earlier commit). This prevents shared new commits based on shared commits from being invalid because the commit on which they are based does not exist in the remote. If the commits contain sensitive information, they should be removed, which involves a more complex procedure to rewrite history.*
* *The git-flow*[*[112]*](https://en.wikipedia.org/wiki/Git#cite_note-114) *workflow and naming conventions are often adopted to distinguish feature specific unstable histories (feature/\*), unstable shared histories (develop), production ready histories (main), and emergency patches to released products (hotfix).*
* *Pull requests are not a feature of git, but are commonly provided by git cloud services. A pull request is a request by one user to merge a branch of their repository fork into another repository sharing the same history (called the upstream remote).*[*[113]*](https://en.wikipedia.org/wiki/Git#cite_note-115) *The underlying function of a pull request is no different than that of an administrator of a repository pulling changes from another remote (the repository that is the source of the pull request). However, the pull request itself is a ticket managed by the hosting server which initiates scripts to perform these actions; it is not a feature of git SCM.*

<https://en.wikipedia.org/wiki/Git#Conventions>

*Ein* ***Pull Request*** *oder* ***Merge Request*** *bezeichnet in der* [*Versionsverwaltung*](https://de.wikipedia.org/wiki/Versionsverwaltung) *einen (webbasierten) Arbeitsablauf,* [*Quellcode*](https://de.wikipedia.org/wiki/Quellcode)*-Änderungen in Softwareprojekten vorzunehmen. Der Begriff wurde von* [*Onlinediensten*](https://de.wikipedia.org/wiki/Onlinedienst) *zur Versionsverwaltung geprägt und geht auf Operationen der Versionsverwaltungssysteme zurück (git pull bzw. git merge).*

*Ziel eines Pull Requests ist, Änderungen aus einem* [*Branch*](https://de.wikipedia.org/wiki/Abspaltung_(Softwareentwicklung)#Versionskontrollsysteme) *in die eigentliche Quellcode-Basis (*[*Upstream*](https://de.wikipedia.org/wiki/Upstream_(Softwareentwicklung))*) zu übernehmen. Wird ein Pull Request akzeptiert, so spricht man von einem Merge, wird er geschlossen, so spricht man von einem Close.*[*[1]*](https://de.wikipedia.org/wiki/Pull_Request#cite_note-:0-1)[*[2]*](https://de.wikipedia.org/wiki/Pull_Request#cite_note-2)[*[3]*](https://de.wikipedia.org/wiki/Pull_Request#cite_note-3)

<https://de.wikipedia.org/wiki/Pull_Request>

<https://www.greengeeks.com/tutorials/make-a-pull-request-in-git/>

<https://www.delftstack.com/howto/git/difference-of-git-clone-and-git-pull/>

<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request>

commit-ish ./. tree-ish

<https://stackoverflow.com/questions/4044368/what-does-tree-ish-mean-in-git>

https://initialcommit.com/blog/what-is-a-tree-in-git

## === Block 2 git mit github etc

*There are many offerings of Git repositories as a service. The most popular are* [*GitHub*](https://en.wikipedia.org/wiki/GitHub)*,* [*SourceForge*](https://en.wikipedia.org/wiki/SourceForge)*,* [*Bitbucket*](https://en.wikipedia.org/wiki/Bitbucket) *and* [*GitLab*](https://en.wikipedia.org/wiki/GitLab)*.*[*[88]*](https://en.wikipedia.org/wiki/Git#cite_note-Krill_2016-89)[*[89]*](https://en.wikipedia.org/wiki/Git#cite_note-Alexa_github.com-90)[*[90]*](https://en.wikipedia.org/wiki/Git#cite_note-Alexa_sourceforge.net-91)[*[91]*](https://en.wikipedia.org/wiki/Git#cite_note-Alexa_bitbucket.org-92)[*[92]*](https://en.wikipedia.org/wiki/Git#cite_note-Alexa_gitlab.com-93)

<https://en.wikipedia.org/wiki/Git#Git_server_as_a_service>

<https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities>

### ==== GitHub & GitLab

***GitHub*** *ist ein* [*netzbasierter Dienst*](https://de.wikipedia.org/wiki/Onlinedienst) *zur* [*Versionsverwaltung*](https://de.wikipedia.org/wiki/Versionsverwaltung) *für* [*Software*](https://de.wikipedia.org/wiki/Software)*-Entwicklungsprojekte. Namensgebend war das Versionsverwaltungssystem* [*Git*](https://de.wikipedia.org/wiki/Git)*. Das Unternehmen GitHub, Inc. hat seinen Sitz in* [*San Francisco*](https://de.wikipedia.org/wiki/San_Francisco) *in den* [*USA*](https://de.wikipedia.org/wiki/USA)*. Seit dem 26. Dezember 2018 gehört das Unternehmen zu Microsoft.*

<https://de.wikipedia.org/wiki/GitHub>

<https://www.oreilly.com/library/view/github-eine/9781098128739/>

<https://docs.github.com/en/get-started/quickstart> official documentation quickstart guide EN

<https://docs.github.com/de/get-started/quickstart> offizielle Dokumentation DE \*\*\*

SSH mit github

remember ssh-keygen etc.

[source, bash, linenums]

----

**$ cat ~/.ssh/id\_rsa.pub**

$ # copy and paste in github account -> profile -> settings -> SSH und GPG

$ Test der SSH Github authentification

**$ ssh -T git@github.com**

**$ ssh -T git@github.com -v # in case of problems**

----

SSH Host Keys Fingerprints

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/githubs-ssh-key-fingerprints>

<https://devconnected.com/how-to-clone-a-git-repository/>

**$ cd ~/git\_kurs**

**$ git clone** [**git@github.com**](mailto:git@github.com)**:<account>/<reponame>.git**

TODO geht damit auch der Push? Ja! Achtung: das geht natürlich nicht mit der Methode https.

**Übungsaufgabe**

Bitte löschen sie im github Account unseren ersten Versuch (**tester**). Legen sie nun in github ein neues Repository mit dem Namen **my\_cool\_project** an. Wechseln sie auf ihrem lokalen Rechner in das Verzeichnis **~/git\_kurs** und clonen sie ihr neues Repository mit der Methode SSH dorthin. Wechseln sie in das geclonte Repository und machen einen ersten commit. Kontrollieren sie über die github Webseite, ob ihr lokaler commit dort angekommen ist.

Zeitrahmen 15 Minuten. Wir sehen uns um 14:15

<https://www.digitalocean.com/community/tutorials/how-to-push-an-existing-project-to-github> das kann man sich sparen, weil bei der Anlage eines neuen Repositories alles genau von github erklärt wird.

**Übungsaufgabe**

Bitte denkt euch ein eigenes Projekt in Java aus. Legt dazu einen Ordner im Verzeichnis **~/git\_kurs/** an. In diesem Ordner mit git init ein Repository anlegen. Hier macht ihr ein paar Commits für euer Projekt. Seid gerne kreativ, es muss auch nicht Java sein. Morgen sehen wir, wie wir das in github importieren können.

***GitLab*** *ist eine* [*Webanwendung*](https://de.wikipedia.org/wiki/Webanwendung) *zur* [*Versionsverwaltung*](https://de.wikipedia.org/wiki/Versionsverwaltung) *für Softwareprojekte auf* [*Git*](https://de.wikipedia.org/wiki/Git)*-Basis, hinzu kamen später weitere Funktionen zur Softwareentwicklung. GitLab bietet ein* [*Issue-Tracking-System*](https://de.wikipedia.org/wiki/Issue-Tracking-System) *mit* [*Kanban-Board*](https://de.wikipedia.org/wiki/Kanban-Tafel)*, ein System für* [*Continuous Integration*](https://de.wikipedia.org/wiki/Kontinuierliche_Integration) *und* [*Continuous Delivery*](https://de.wikipedia.org/wiki/Continuous_Delivery) *(CI/CD), ein* [*Wiki*](https://de.wikipedia.org/wiki/Wiki)*, eine* [*Container*](https://de.wikipedia.org/wiki/Containervirtualisierung)*-Registry, einen Sicherheitsscanner für Container und Sourcecode sowie Multi-Cluster-Verwaltung und -Überwachung. GitLab ist in Produkte für Entwickler integrierbar und selbst über eine API fernsteuerbar. GitLab ist in den* [*Programmiersprachen*](https://de.wikipedia.org/wiki/Programmiersprache)[*Ruby*](https://de.wikipedia.org/wiki/Ruby_(Programmiersprache)) *und* [*Go*](https://de.wikipedia.org/wiki/Go_(Programmiersprache)) *entwickelt.*

*Die GitLab Community Edition (CE) wird als* [*Open-Source*](https://de.wikipedia.org/wiki/Open_Source)*-Software unter der* [*MIT-Lizenz*](https://de.wikipedia.org/wiki/MIT-Lizenz) *entwickelt. Seit August 2013 bietet die GitLab Inc. für Unternehmen relevante Funktionen in der Enterprise Edition (EE) an.*[*[3]*](https://de.wikipedia.org/wiki/GitLab#cite_note-3) *Man kann GitLab auf eigener Hardware betreiben oder seit 2012 auf GitLab.com als GitLab Enterprise Edition – als* [*Software as a Service*](https://de.wikipedia.org/wiki/Software_as_a_Service) *(SaaS). Neben kostenlosem Hosting privater und öffentlicher* [*Repositories*](https://de.wikipedia.org/wiki/Repository) *wird auch kostenpflichtiger Support angeboten.*[*[4]*](https://de.wikipedia.org/wiki/GitLab#cite_note-4) *GitLab ist damit eine Alternative zu ähnlichen Angeboten wie* [*GitHub*](https://de.wikipedia.org/wiki/GitHub) *und* [*Bitbucket*](https://de.wikipedia.org/wiki/Bitbucket)*.*

<https://de.wikipedia.org/wiki/GitLab>

<https://www.heise.de/news/Streit-zwischen-Open-Source-Community-und-GitHub-eskaliert-7160074.html>

<https://www.heise-events.de/workshops/gitlab>

gitlabracadabra.deb

gitlab-cli.deb

<https://confluence.jaytaala.com/display/TKB/Gitlab+technical+install>

### ==== git Server

*As Git is a distributed version control system, it could be used as a server out of the box. It's shipped with a built-in command* **git daemon** *which starts a simple TCP server running on the GIT protocol.*[*[81]*](https://en.wikipedia.org/wiki/Git#cite_note-82) *Dedicated Git HTTP servers help (amongst other features) by adding access control, displaying the contents of a Git repository via the web interfaces, and managing multiple repositories. Already existing Git repositories can be cloned and shared to be used by others as a centralized repo. It can also be accessed via remote shell just by having the Git software installed and allowing a user to log in.*[*[82]*](https://en.wikipedia.org/wiki/Git#cite_note-83) *Git servers typically listen on* [*TCP port*](https://en.wikipedia.org/wiki/Port_(computer_networking)) *9418.*[*[83]*](https://en.wikipedia.org/wiki/Git#cite_note-84)

<https://en.wikipedia.org/wiki/Git#Git_server>

<https://git.sdf.org/>

<https://docs.gitea.io/en-us/install-from-binary/>

## === Block 3 Grundlagen von git mit GUI (Visual Studio - Code) (optional)

Der dritte Block ist optional, aber zur Vertiefung und Wiederholung des Stoffes

empfohlen. Die Woche kann unter Linux oder MS-Windows gemacht werden.

### ==== Installation Visual Studio Code

Installation von Visual Studio - Code unter Linux (oder MS-Windows).

Plugins:

gitlens (siehe Kofler)

<https://ardalis.com/git-graph-visualizes-branches-in-vs-code-for-free/>

### ==== Stoffwiederholung mit IDE: Visual Studio Code

**Übungsaufgabe**

Installieren sie das Debian Paket Visual Studio Code nach dieser Anleitung.

<https://code.visualstudio.com/docs/setup/linux>

Visual Studio Code kann aus der GUI gestartet werden. Oder über das CLI:

code --help

Probieren sie die bisher gemachten HTML Beispiele zu git mit Visual Studio Code.

<https://code.visualstudio.com/docs/languages/html>

<https://code.visualstudio.com/docs/sourcecontrol/intro-to-git> \*\*

<https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-de> \*\* kurz und gut, auch mit Verbindung per SSH zu github (auch in englischer Sprache).

Zeitrahmen: 60 Minuten

**Übungsaufgabe**

Programmiert XSS or XRSF in HTML. Braucht das ein Blue-Team und ein Red-Team?

Zeitrahmen: 30 Minuten

## === Block 5 Continuous Integration (Jenkins)

<https://www.jenkins.io/doc/pipeline/tour/getting-started/>

*Gitlint checks your commit messages for style. Great for use as a* [*commit-msg git hook*](https://jorisroovers.com/gitlint/latest/commit_hooks/) *or as part of your gating script in a* [*CI pipeline*](https://jorisroovers.com/gitlint/latest/ci/)*.*

<https://jorisroovers.com/gitlint/latest/> \*\*\*

apt install gitlint # Debian 12

# == Kursabschluss

Was sollten sie behalten haben:

git add

git commit

git clone

git pull

git push

git cheat sheet (in verschiedenen Sprachen)

<https://training.github.com/>

Als Alternative zum cheat sheet **git help | less**

Wer wird in Zukunft git verwenden?

# == Anhang

## === Literatur (Bibliography)

Kofler:

<https://www.rheinwerk-verlag.de/git_5478/> als Lehrbuch für Anfänger nicht geeignet. Experten finden interessante Hinweise.

Sascha Kersken: IT-Handbuch, 10. Auflage 2021

<https://www.rheinwerk-verlag.de/it-handbuch-fuer-fachinformatiker-der-ausbildungsbegleiter/> das Thema findet sich dort nicht, oder?

In Auflage 9 ist es Kap. 12.2

Offizielles git Lehrbuch:

Als Lehrbuch für Anfänger nicht geignet. Zur Vertiefung für Experten geeignet. Auch in deutsch (offizielle Dokumentation). <https://git-scm.com/book/de/v2> Viele interessante Details, ein paar gute Schaubilder. Fängt mit zu vielen technischen Details an. Viele Beispiele sind nur bruchstückhaft nachvollziehbar. Veraltet, von 2014. Trotzdem empfohlen. Aber nicht als Einstieg. Als online Dokument frei verfügbar.

Rene Preißel, Björn Stachmann: git dpunkt Verlag 2019 5. Auflage (Google Books)

Die ersten 14 Kapitel decken auf deutsch und etwa 100 Seiten die kompletten LPIC 701 Lernziele ab, sind brauchbar geschrieben. Ich habe danach alles verstanden. Nichtsdestowenigermehr ist es didaktisch nicht so, wie ich es gerne hätte (es fehlt mir ein von Anfang an durchgängiges Beispiel). Beim zweiten Lesen gefällt es mir nicht mehr so gut. Und beim dritten Lesen noch schlechter. Schaubilder sind relativ gut, aber es fehlen zu viele Details. Auch die Kommandos werden nur halb genannt, wichtige Details zum Funktionieren oder Verstehen fehlen. Beispiel: Tag und Signing kann so nicht funktionieren. Es gibt kaum konkrete Beispiele. Leider häufige Verwendung von foo und bar. Gut am Ende des Buches ein Kapitel zu Jenkins, aber auch das bleibt im Ungefähren.

<https://dpunkt.de/produkt/git/> Google Books? 3. Auflage Libby 5. Auflage

Gilbert Fongan

<https://www.lpi.org/de/blog/2023/07/11/boost-your-it-career-by-learning-devops-with-lpi/>

<https://gitlab.com/GilbertFongan/devops-book-labs/>

<https://gitlab.com/GilbertFongan/devops-book-labs/-/raw/main/PRACTICAL%20DEVOPS%20TOOLS.pdf> \*\*\* Schaubilder !!! MODULE I-3 : Source Code Management, page 75 - 103

Paul Fuchs: Git Handbuch für Einsteiger: Der leichte Weg zum Git-Experten (Taschenbuch), auch als eBook und Bundle. Sieht brauchbar aus (Screenshots unter Windows!). Im Kindle Abo enhalten kostenlos.

<https://bmu-verlag.de/books/git/>

## === todo list: githowto

Als Kursmaterial für LPIC 701 geeignet (ein durchgängiges HTML Beispiel) <https://githowto.com/> Es fehlen Schaubilder. Erklärungen sind zu kurz. Veraltet: master -> main, Verwendung von checkoutb beim undo, anstelle von restore. Ein brauchbarer Fahrplan in kleinen Schritten fast alles drin (add, commit, checkout, branch, push, pull, clone) und in überschaubarer Zeit machbar (halber Tag bis eine Woche) !!! Es fehlt Verwendung von Submodules und gitignore.

change master to main

change deprecated git subcommands

colorize output

## === todo list: misc

URLs der manpages von Debian auf offizielle SCM Seite umstellen.

Unterschied: Head - HEAD Kofler Kapitel 3

$ aptitude search "^git"

<https://stackoverflow.com/questions/2221658/whats-the-difference-between-head-and-head-in-git>

<https://www.golinuxcloud.com/git-head-caret-vs-tilde-at-sign-examples/>

<https://initialcommit.com/blog/what-is-git-head>

<https://stackoverflow.com/questions/8201765/how-to-get-list-of-branch-heads-in-git>

## === Wiederholung und Vertiefung des Unterrichtsmaterials

**Übungsaufgabe**

*Githug is designed to give you a practical way of learning git. It has a series of levels, each requiring you to use git commands to arrive at a correct answer.*

<https://github.com/Gazler/githug>

[source, bash, linenums]

----

**# apt install ruby-full**

**# gem install githug**

----

[source, bash, linenums]

----

**$ mkdir ~/git\_kurs/githug**

**$ cd ~/git\_kurs/githug**

**$ githug help**

----

Die ersten beiden Level spielen wir zusammen. Vielleicht auch das dritte Level. Aktuell gibt es 56 Level.

Quelle: Kofler

<https://www.w3schools.com/git/git_exercises.asp>

<https://training-course-material.com/training/Git_exercises>

<https://coderefinery.github.io/git-intro/exercises/>

## === Alternative Unterrichtskonzepte

$ git help everyday # not helpful WH

<https://www.linuxhotel.de/course/git-de/> 2 Tage git im Linuxhotel

<https://www.biteinteractive.com/picturing-git-conceptions-and-misconceptions/> \*\*\* MUST READ

<https://manpages.debian.org/bookworm/git-man/gitcore-tutorial.7.en.html> \*\*

<https://www.atlassian.com/git/tutorials> \*\*\*

<https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/> Hinweis: master heisst in der neuen political correctness main.

<https://coderefinery.github.io/git-intro/> 1-2 day course

21.3 Versionsverwaltung des Quelltextes 1020

21.3.1 Einführung in die Versionsverwaltung 1020

21.3.2 Das Versionsverwaltungssystem Git installieren und konfigurieren 1024

21.3.3 Ein neues lokales Repository anlegen 1026

21.3.4 Ein bestehendes Repository klonen 1026

21.3.5 Änderungen in den Staging-Bereich übertragen 1027

21.3.6 Änderungen in das lokale Repository übertragen 1027

21.3.7 Die verschiedenen Zustände in Git 1029

21.3.8 Änderungen in das Remote Repository übertragen 1030

21.3.9 Änderungen aus dem Remote Repository übertragen 1031

21.3.10 In einem neuen Branch arbeiten 1032

21.3.11 Änderungen aus einem Branch übernehmen 1033

21.3.12 Übersicht über die wichtigsten Befehle und Begriffe 1034

<https://www.rheinwerk-verlag.de/javascript-das-umfassende-handbuch/>

Philip Ackermann: JavaScript. Das umfassende Handbuch 3. Auflage (oder die 2.??)

<https://www.coursera.org/learn/version-control-with-git> nur audit ohne Zertifikat ist kostenlos?

## === asciidoc

:author: Werner Heuser

:email: werner.heuser@web.de

:revnumber: v0.3

:revdate: 23.07.2023

:title: Basiswissen Versionskontrolle mit git

:description: Eine kurze Einführung in git mit Beispielen.

:doctype: book

:chapter-label:

:sectnums:

:toc: left

:toclevels: 3

:toc-title: Inhaltsverzeichnis

//:front-cover-image: image::images/cover.jpg[]

:source-highlighter: pygments

:pygments-linenums-mode: table // tut nicht, bleibt inline

//:pygments-style: manni

// <https://pygments.org/>

//:source-highlighter: coderay

//:coderay-linenums-mode: inline

//:coderay-linenums-mode: table

//:nofooter:

////

todo: source code mit table nummerierung an stelle

von inline, das macht copy and paste schwieriger und/oder

unsichtbare bugs in code einbauen, z.B.

<https://invisibletext.com/>

////

////

**$ asciidoctor -v -o basiswissen\_git.html -b html5 basiswissen\_git.adoc**

<https://marketplace.visualstudio.com/items?itemName=asciidoctor.asciidoctor-vscode>

<https://mrduguo.github.io/asciidoctor.org/docs/asciidoc-syntax-quick-reference/>

<https://marketplace.visualstudio.com/items?itemName=flobilosaurus.vscode-asciidoc-slides>

<https://www.heise.de/news/Alles-auf-Markdown-Mozilla-Dokumentation-legt-HTML-ad-acta-7259525.html>

<https://www.alexdico.com/asciidoctor/2022/03/08/efficient-writing-with-asciidoctor.html> \*\*\*

Conversion from LibreOffice to Asciidoc

https://github.com/unoconv/unoserver/

Write in asciidoc but render with asciidoctor!

~/.vimrc

autocmd filetype asciidoc set shiftwidth=2

" filetype indent on

" set smartindent

////

<https://asciidocfx.com/>

<https://www.linux-community.de/ausgaben/linuxuser/2020/10/asciidocfx-vereinfacht-das-schreiben-von-asciidoc-dokumentationen/>

## === FAQ zu diesem Dokument

**Gibt es dieses Dokument mit git und asciidoc?**

Nein, noch nicht. Google Docs hat eine Versionskontrolle. Später vielleicht ein hybrider Ansatz: asciidoc und Google Docs (oder besser CryptPad) plus git.

**Ist dieses Dokument schon fertig?**

Nein. Early Development.

////

# == Inhaltsverzeichnis

[**Protokoll 1**](#_fj4mbf1lxnnq)

[**== Über dieses Dokument 2**](#_ud36el2r8vdz)

[**== Über den Dozenten 3**](#_68hjuaospknc)

[**== Konzept: Warum git? Welche Zielgruppen? Kursziele 4**](#_uj9lw54jearp)

[=== Warum git für Anwendungsentwickler? 4](#_bgyy4ewj9egs)

[=== Warum git für Systemintegratoren? 4](#_elj8ha9jkikn)

[=== Wie ist git entstanden? 4](#_moij4egcz1y1)

[=== Warum Linux als Basis für diesen Kurs? 4](#_almkzj77jir1)

[=== Warum auf der Kommandozeile? 4](#_10w5nlulohm1)

[=== Ziele des Kurses 5](#_wlqb17fhorqz)

[=== Nebenziele des Kurses 5](#_20997sgrbe8i)

[**== Kursplan 6**](#_vpvwafb2zozw)

[=== Block 1 Grundlagen von git auf der Kommandozeile 6](#_441vxdj4w7f6)

[==== Intro 6](#_ku5q392zg7k1)

[==== Wiederholung Linux 6](#_kyd6c2qf76w)

[==== Das Kommando diff 8](#_q6n23sxs4pox)

[==== Vergleich einer grösseren Datei 10](#_tlijt9jbeaqw)

[==== Übungsaufgaben 12](#_32x9gf9r1zte)

[==== HTML & CSS Basics 15](#_vofvzjbz3ezr)

[==== HTML & CSS Validation 18](#_gry6eg3nh6wy)

[==== Was ist Versionskontrolle? 20](#_jlhtvulqiqse)

[=== Preparation: Installation and Global Configuration 22](#_73tsoesqvk2j)

[==== Installation git githowto 0 22](#_hxno0carjqtu)

[==== Global Configuration githowto 1 Preparation 23](#_5hbpn0qqiyxu)

[==== Download git howto stuff githowto 2 The final preparation 25](#_v3x44g3utupl)

[=== Creating a First git Project 26](#_8eb0kd28n58l)

[==== Konfiguration eines ersten git Repository githowto 3 Creating a project 26](#_b8vgf56o2ynd)

[==== githowto 4 Checking the status of the project 28](#_gud6alx1gpxa)

[==== githowto 5 Making changes 29](#_bhb8iej6uwgn)

[==== githowto 6 Staging the changes 30](#_8x7ddic6np0v)

[==== githowto 7 Staging and commiting [Why make it complicated?] 31](#_m9j2eoixiapk)

[==== githowto 8 Committing the changes 34](#_rz5aib2n1ewr)

[==== Änderungen werden verwaltet, und nicht Dateien githowto 9 Changes, not files 35](#_faap3mxncfkm)

[==== githowto 10 History 39](#_tmadn9uxjav1)

[==== Was haben wir erreicht A? git funktioniert schon das kann man mit git log und git diff sehen 40](#_3yuin3yijh3d)

[==== Exkurs: Wie findet man die git man pages? 42](#_cbqmgmi9m1u2)

[==== Änderungen im Detail: working tree -> stage (aka index) -> repository (local) 43](#_1tyddwockf0u)

[==== githowto 11 Aliases 44](#_oh7rs8fnmzdg)

[==== git checkout - githowto 12 Getting older versions 45](#_zg27px2t72kl)

[==== Was haben wir erreicht B? time travel funktioniert schon: git checkout 46](#_kj85yfah9ab3)

[==== Tags githowto 13 Tagging versions 47](#_go327ntc64a9)

[=== How to Handle Mistakes 49](#_vnvnzegw2zu3)

[==== githowto 14 Discarding local changes 49](#_54cmp7ssv88p)

[==== githowto 15 Cancel Staged changes (before committing) 50](#_r6xmrdn85kd6)

[==== githowto 16 Cancelling commits 51](#_kaia2bhj3oko)

[==== githowto 17 Removing a commit from a branch 52](#_w22y76ek9vvk)

[==== githowto 18 Removing the oops tag 55](#_ej2lhp8zvm44)

[==== githowto 19 Changing commits 56](#_xq2kk39v7cl)

[==== githowto 20 Moving files 58](#_98mdbtttbcv1)

[=== Inside the git Machine: .git and git objects 59](#_68vw8ywmuv86)

[==== githowto 21 More information about the structure 59](#_dybeqi6abm89)

[==== githowto 22 Inside git: .git directory 60](#_4p2x1eoy3tzs)

[==== githowto 23 Git inside: Direct work with git objects 62](#_4ihwrcr5csjp)

[=== Working with Branches 63](#_7ij2zd3k399o)

[==== Ein erster Branch githowto 24 Creating a branch 63](#_m4ws44k1v645)

[==== githowto 25 Navigating branches 64](#_6yk0vvuok96g)

[==== githowto 26 Changes to master (now main) branch 67](#_6g4u3vqjlg0f)

[==== githowto 27 View the different branches 68](#_n54s4aekxajq)

[=== Merging Branches 71](#_xcwiase8cf1q)

[==== Zusammenführung von branches (ohne Konflikt) githowto 28 Merging 71](#_ezbtwi4mfinp)

[==== githowto 29 Creating a conflict 72](#_wdlft3xvlmuj)

[==== githowto 30 Resolving conflicts 82](#_hc38ol9blyu4)

[=== Rebasing Branches (Preparation) 83](#_lkh74aog01u3)

[==== githowto 31 Relocating as an alternative to merging 83](#_tsooxvg6m43k)

[==== githowto 32 Resetting the style branch 85](#_43oyl15ddlnu)

[==== githowto 33 Reset of the master [now main] branch 86](#_62l1kboc4ycx)

[=== Rebasing Branches 87](#_3g88uctweyue)

[==== githowto 34 Rebase 87](#_p29oxwir6m7k)

[==== githowto 35 Merging to the master (now main) branch 88](#_zgt76i48m0c2)

[=== Multiple Repositories: clone, push, pull 89](#_ye1n0c155560)

[==== githowto 36 Multiple Repositories 89](#_6aw1wokpav17)

[==== githowto 37 Cloning repositories 90](#_k9bqh3fgs5yp)

[==== githowto 38 Examine the cloned history 91](#_69i6wq100te5)

[==== githowto 39 What is origin? 92](#_h8abr0n0d4r0)

[==== githowto 40 Remote branches 93](#_9jh5x39horco)

[==== githowto 41 Changing the original repository 94](#_o85sect3bl7o)

[==== githowto 42 Fetching changes 95](#_uqveo82zwenr)

[==== githowto 43 Merging pulled changes 96](#_v2wozm3337bm)

[==== githowto 44 Pulling and merging changes 97](#_u24jqq1poaah)

[==== githowto 45 Adding a tracking branch 98](#_7jhj7bfhu29l)

[=== Bare Repos 99](#_boms1fnakk5y)

[==== githowto 46 Bare repos 99](#_c9x7cfffoq57)

[==== githowto 47 Adding a remote repository 101](#_q2w57f3mkk2o)

[==== githowto 48 Submitting changes 102](#_yo6cwo6wz7tn)

[==== githowto 49 Removing common changes 103](#_8kx3mnhl839p)

[==== githowto 50 Placing your git repository 104](#_5o37xzvgzon5)

[==== githowto 51 Sharing repositories 105](#_1yyragfb9xu4)

[==== githowto Thank you! 106](#_f2yy9h8h51un)

[=== Beyond the githowto: 106](#_qux4zyjd1zzc)

[==== git fsck 107](#_6xnn4ycu32ua)

[==== git gc - garbage collection 107](#_qbac6ycroeba)

[==== submodules 107](#_h7jruga52cq8)

[==== .gitignore 108](#_hsmua37j0rex)

[==== git stash 109](#_dooovr1arwcg)

[==== git worktree 109](#_3xgpdidzfl9b)

[==== undo 109](#_kggs1ggcmrau)

[==== FAQ, Tipps and Tricks 110](#_8mtikmoswwod)

[==== Glossar 111](#_6rleztdtslpa)

[=== Block 2 git mit github etc 113](#_8eomxfl79nhc)

[==== GitHub & GitLab 113](#_inylzyng7myf)

[==== git Server 115](#_wfy3osvff9j7)

[=== Block 3 Grundlagen von git mit GUI (Visual Studio - Code) (optional) 116](#_31n0lqatavfo)

[==== Installation Visual Studio Code 116](#_f5u2xu30ouvm)

[==== Stoffwiederholung mit IDE: Visual Studio Code 116](#_hit9z9ty87ea)

[=== Block 5 Continuous Integration (Jenkins) 117](#_pzm03n6pi4u8)

[**== Anhang 118**](#_iwvukjii9ez2)

[=== Literatur (Bibliography) 118](#_udrq81wd6he)

[=== todo list: githowto 119](#_4cuv3wbynida)

[=== todo list: misc 119](#_2vovifn5onj0)

[=== Wiederholung und Vertiefung des Unterrichtsmaterials 119](#_8a60a297sder)

[=== Alternative Unterrichtskonzepte 121](#_3qnxmsl5m9u)

[=== asciidoc 122](#_jyqxb2ptwehw)

[**=== FAQ zu diesem Dokument 123**](#_htvrp0whoiyo)

[== Inhaltsverzeichnis 124](#_urgdhpsh6oz8)

////