#### **Begriffe zu Workflow**

* **central repository:** Der Speicherort des Archivs. Hier liegen alle von der Software verwendeten Daten zu einem Projekt.

Wie eine Bücherei. Man schreibt ein neues Buch und stellt es ins Regal. Wenn man es aktualisieren möchte kann man es nicht einfach verbessern und neu reinstellen und das Alte raus nehmen.

* **repository:** is the most basic element of GitHub. They're easiest to imagine as a project's folder. A repository contains all of the project files (including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private.
* **Workflow:** Festlegen der Aufgaben, Verarbeitungseinheiten, sowie deren Beziehungsgeflecht innerhalb des Prozesses (z.B. Datenfluss).

= Arbeitsablauf

#### **git-flow:** ist eine git-Erweiterung von Vincent Driessen, die die Umsetzung seines Branching-Modells deutlich vereinfacht.

* **Bit Keeper:** Software-Werkzeug zur Versionsverwaltung von Software-Quelltext
* **RCS:** Abkürzung für "Revision Control System" oder auf Deutsch: Versionsverwaltungssystem.
* **Versionsverwaltung:** System zur Erfassung von Änderungen an Dokumenten oder Dateien. Alle Versionen werden in einem Archiv mit Zeitstempel und Benutzererkennung gesichert und können später wiederhergestellt werden. Versionsverwaltungssysteme werden typischerweise in der Softwareentwicklung eingesetzt um Quellcode zu verwalten.
* **SVN, CVS:** Abkürzungen zweier weit verbreiteter Versionsverwaltungssysteme: Subversion und Concurrent Version System.
* **Git:** eine Versionsverwaltung für Dateien, ähnlich SVN und CVS. Der Fokus von Git liegt dabei auf der verteilten Struktur, bei der sich jeder Benutzer eine Kopie des Repositories anlegt, in seiner Kopie Änderungen macht und dann einen sogenannten Pull Request schickt, damit die Änderungen in das Haupt-Repository übernommen werden.

#### Freie Software zur verteilten Versionsverwaltung von Dateien: Zusammenschluss unabhängiger Computer, die sich für den Benutzer als ein einziges System präsentieren. Nach Peter Löhr: „ eine Menge interagierender Prozesse (oder Prozessoren), die über keinen gemeinsamen Speicher verfügen und daher über Nachrichten miteinander kommunizieren“.

#### Wird auch zum Speichern flach strukturierter Datensätze direkt als Datei genutzt.

* **Commit:** Ausdruck aus der [Softwaretechnik](https://de.wikipedia.org/wiki/Softwaretechnik), der die bestätigende [Freischaltung](https://de.wikipedia.org/wiki/Freischaltung) einer Änderung beschreibt. Er wird sowohl im Zusammenhang mit der [Persistierung](https://de.wikipedia.org/wiki/Persistenz_%28Informatik%29) von Daten in einer Datenbank als auch beim Einpflegen von Programm-Quelltext in [Versionsverwaltungssystemen](https://de.wikipedia.org/wiki/Versionsverwaltung) verwendet.

Als *Commit* oder *Check-In* bezeichnet man bei der Verwendung von Versionsverwaltungssystemen den Vorgang des Einspielens von neuem oder geändertem Quelltext bzw. anderen Dateien in das Versionsverwaltungssystem. Dabei wird eine neue Version der Software den anderen an der Softwareentwicklung beteiligten Entwicklern zur Verfügung gestellt.

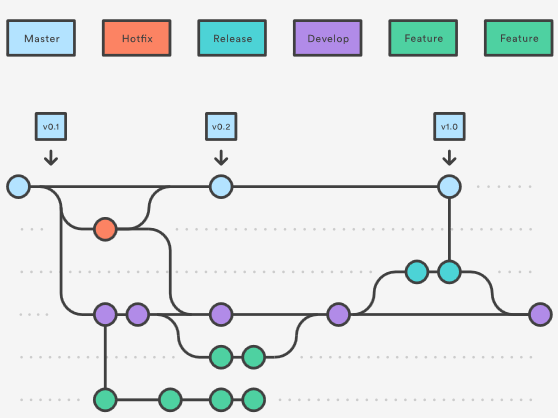
* **commit, or "revision":** is an individual change to a file (or set of files). It's like when you *save* a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.
* **Revision:** Ein Entwicklungszustand einer Software oder allgemein einer Datei.
* **Tag:** Wichtige Revision, der ein besonderer Name gegeben wurde, damit man schnell auf sie zugreifen kann. Beispiele aus dem phpBB Repository sind release-3.0.11, release-3.0.10, usw., die die jeweils veröffentlichte Version markieren)
* **Branch:** zu deutsch Zweig, ist eine Abspaltung von einer anderen Version, so dass unterschiedliche Versionen parallel weiterentwickelt werden können. Änderungen können dabei von einem Branch auch wieder in einen anderen einfließen, das wird dann als Merging, zu deutsch verschmelzen, bezeichnet. Oft wird der Hauptentwicklungszweig als Trunk (z. B. bei [Subversion](https://de.wikipedia.org/wiki/Apache_Subversion)) oder Master (z. B. bei [Git](https://de.wikipedia.org/wiki/Git)) bezeichnet.
* **branch:** is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.
* **Master-Branch:** „Standard“-Branch, wenn man ein neues Repository (verwaltendes Verzeichnis zur Speicherung und Beschreibung digitaler Objekte) erstellt. Man sollte aber für die Entwicklung andere Branches verwenden und diese dann in den Master-Branch zusammenführen (mergen).
* **Hooks:** Möglichkeit eigene Skripte zu starten, wenn bestimmte, wichtige Ereignisse auftreten. Es gibt zwei Gruppen dieser Einschubmethoden: Hooks für den Client und Hooks für den Server. Die Hooks für den Client können bei Ereignissen, wie zum Beispiel einem Commit oder Merge, eingerichtet werden. Die Hooks für den Server können bei Operationen wie den Empfang von hochgeladenen Commits, ausgeführt werden. Es gibt viele Möglichkeiten diese Hooks sinnvoll einzusetzen. Einige davon werde ich hier vorstellen.
* **Pull:** Pull refers to when you are fetching in changes and merging them. For instance, if someone has edited the remote file you're both working on, you'll want to pull in those changes to your local copy so that it's up to date.
* **Push:** Pushing refers to sending your committed changes to a remote repository such as GitHub.com. For instance, if you change something locally, you'd want to then push those changes so that others may access them.
* **Remote Repositorys (Externe Repositorys):** sind Versionen meines eigenen Projektes, die im Internet oder irgendwo in einem anderen Netzwerk gespeichert sind. Man kann mehrere solcher Repository haben. Um mit anderen zusammen arbeiten zu können, muss man solche Repositorys verwalten, Daten aus ihnen herunter- oder heraufladen, um die Arbeit für andere verfügbar zu machen.

**Gitflow Workflow**

* Legt ein strenges Verzweigungsmodell um die Versionen des Projektes fest
* Bietet einen soliden Rahmen für die Verwaltung größerer Projekte
* Ordnet spezifische Rollen den verschiedenen Verzweigungen zu und definiert, wie und wann sie miteinander interagieren sollen.
* Verwendet individuelle Zweige für vorbereitende, unterstützende und aufgezeichnete Versionen
* Außerdem: beinhaltet Anfragen, isolierte Experimente und eine effiziente Zusammenarbeit
* Verwendet zentralen Verwaltungsort als Kommunikations-hub für alle Entwickler
* Entwickler arbeiten lokal und legen ihre fertigen Zweige in die zentrale Verwaltung

* Besonderheit hier ist die Zweigstruktur des Projektes:

Anstatt eines „master-branch“ benutzt dieser Workflow zwei Zweige zum Speichern der „Projektgeschichte“ (älteren Versionen).



* Master-Branch: speichert die offizielle, veröffentlichte „Geschichte“ (die Versionen)
* Develop-Branch: dient als Integrations-Branch für Funktionen (Features) und Eigenschaften
* Es ist auch gebräuchlich alle Commits in den Master-Branch mit einer Versionsnummer zu tuen.
* Feature-Branches: jedes neue Feature besitzt einen eigenen Feature-Zweig. Dieser kann für ein Backup oder zur Zusammenarbeit in das zentrale Repository geschoben werden. Anstatt des Master-Branch verwenden Feature-Branch den Develop-Branch als Eltern-Zweig und zweigen von ihm ab. Ist das Feature komplett verschmilzt es mit dem Develop-Zweig. Features sollten nie direkt mit dem Master-Branch interagieren.
* Release-Branch: Wird vom Develop-Branch abgespalten, wenn genügend neue Features für eine Freigabe vorhanden sind.

→ Start des Freigabe-Zyklus: Es können keine neuen Features mehr hinzugefügt werden. Nur noch Fehlerbehebungen, Dokumentationen und andere freigabeorientierte Anwendungen. Ist alles soweit fertig bekommt die Freigabeversion eine Versionsnummer und wird mit dem Master-Branch verschmolzen. Zusätzlich sollte der Release-Branch auch zurück in den Develop-Branch gehen, der evtl. schon weiter bearbeitet wurde, seit dem die Freigabe gestartet wurde.

* Vorteil: Ein Team kann an der Freigabe für eine Version arbeiten und ein anderes Team kann weiter an den Features für die darauf folgende Version arbeiten.
* Maintenance-Branch/ Hotfix-Branch: wird benutzt, um schnell Produktionsfreigaben aus zu bessern. Er ist der einzige Zweig, der direkt vom Master-Branch abzweigt. Sobald die Lösung komplett ist, sollte sie mit dem Master- und Develop-Branch (oder dem aktuellen Release-Branch) verschmelzen und im Master-Branch mit einer neuen Versionsnummer ausgezeichnet werden.
* Vorteil: das Team kann Lösungen für Aspekte hinzufügen, ohne den Rest des Workflows zu unterbrechen oder auf den nächsten Release-Zyklus warten zu müssen.

**Internetquellen:**

<https://rogerdudler.github.io/git-guide/index.de.html>

<https://www.veit-schiele.de/dienstleistungen/technische-dokumentation/git/git-flow>

<https://de.wikipedia.org/wiki/Versionsverwaltung#Branch>

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

<https://de.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

<https://git-scm.com/book/de/v1/Git-individuell-einrichten-Git-Hooks>

<https://git-scm.com/book/de/v1/Git-Grundlagen-Ein-Git-Repository-anlegen>

<https://www.phpbb.de/kb/git_installieren>

C:\Users\VWQ2MTU\AppData\Local\Programs\Git\mingw64\share\doc\git-doc\index.html

https://git-scm.com/book/de/v1/Git-Grundlagen-Mit-externen-Repositorys-arbeiten

**Grundlegende Terminal-Begriffe**

* **directory:** Verzeichnisse werden directory genannt.
* **home directory:** das Arbeitsverzeichnis, in welchem das Terminal startet – in der Regel /Users/Name/.
* Mit Rauf- und Runterpfeiltasten wechseln zwischen den zuletzt eingetippten Befehlen
* Tippen von **cd**, dann die Anfangsbuchstaben des Verzeichnisses und mittendrin die Tabulatortaste, um schnell in Verzeichnisse zu springen.
* **Wiederkehrende Befehlsketten**:

1. Anlegen von Batch-Datei (einfache Textdatei ohne Dateiendung). In jeder Zeile dieser Datei wird ein Befehl kopiert.
2. Damit Datei ausgeführt wird müssen Rechte geändert werden: für die Datei stapelverarbeitung mit chmod-Befehl (chomd 755 stapelverarbeitung) auf 755 stellen.
3. Ausführen der Datei mit ./stapelverarbeitung (./ muss davor stehen!)

* **Farbschemata ändern** mit CMD + , bzw. über das Menü Terminal → Einstellungen → Einstellungen.

Vorher: Datei runterladen, die auf .terminal endet und speichern (evtl. muss man Sicherheitseinstellungen des Rechners kurzzeitig außer Kraft setzten), Doppelklicken in Einstellungen des Terminals geöffnetes Farbschemata als Standard setzen.

* **Verzeichnis mit Inhalt löschen:** rm -rf verzeichnisname
* **Versteckte Ordner und Dateien im Terminal anzeigen:** ls –a

ls listet die Dateien in einem Ordner auf, mit –all bzw. –a werden auch versteckte Dateien angezeigt

* **Verzeichnis als Textdatei speichern:** ls > verzeichnisinhalt.txt

Mit > leitet man due Ausgabe in eine .txt-Datei um.

Mit ls -la > verzeichnisinhalt.txt werden die Dateien als ausführliche Liste mit allen Angaben ausgegeben

* **Öffnen von Dateien:** open verzeichnisinhalt.txt

Das Erstellen und Öffnen der Textdatei kann man auch verketten:

ls > verzeichnisinhalt.txt;open verzeichnisinhalt.txt

* **Bilder zuschneiden mit sips**:

1. Bild anpassen und zurechtschneiden und **Seitenverhältnis ignorieren**:

sips -z 768 1024 image.png

1. Bild anpassen **mit Seitenverhältnis**: sips -Z 480 image.png

* Befehl –out: Bestimmen von neuem Dateinamen oder einem Ordner, in den die bearbeiteten Dateien gespeichert werden.
* **Dateien in Ordner thumbnails speichern**: sips -Z 150 -c 150 150 \*.jpg --out thumbnails
* Sips Programm von Apple, Befehl nur für Apple-Betriebssystem
* **Sonderzeichen eingeben:**
* Tilde ~ mit Tastenkombination Alt + n und anschließend Leertaste drücken
* Backslash \ mit Umschalten + Alt + 7
* Geschwungene Klammern { und } mit Alt + 8 oder 9

|  |  |  |
| --- | --- | --- |
| Die wichtigsten Terminal-Befehle | | |
| cal |  | Zeigt den Kalender an |
| cat <datei> |  | Gibt <datei> aus. |
| cd <Verzeichnisname> |  | Wechselt in ein Verzeichnis tiefer |
| cd .. |  | Wechselt in ein höheres Verzeichnis |
| cd / |  | Wechselt in die oberste Ebene der Festplatte, das sogenannte Root- oder Wurzelverzeichnis (nicht zu verwechseln mit dem Benutzerverzeichnis des ebenfalls "Root" genannten Superusers von [MacOSX](http://www.apfelwiki.de/Main/MacOSX)  Beispiel: cd /usr/local/phoenix |
| cd ~ |  | Wechselt in das eigene Benutzerverzeichnis |
| cd |  | ohne Parameter wechselt in das eigene Benutzerverzeichnis |
| cp quelle ziel |  | Kopiert eine Datei quelle nach ziel. Dabei kann ziel ein neuer Name sein (dann wird die Datei sozusagen im Flug umbenannt) oder ein Verzeichnis oder eine existierende Datei - diese würde allerdings überschrieben! |
| clear |  | Löscht den Terminalinhalt, Shortcut CTRL-L (^L) |
| date |  | Zeigt das Datum an |
| dir |  | Prüft, ob das Dokument bereits vohanden ist.  Beispiel: $ dir README.rdoc\  -bash: dir: command not found\ |
| [find](http://www.apfelwiki.de/Main/Find) |  | Suchen von Dateien nach div. Kriterien |
| grep |  | Durchsucht die Eingabe nach einem Suchausdruck und gibt gefundene Zeilen zurück |
| less dateiname |  | Zeigt den Inhalt der Datei *dateiname* an. |
| locate |  | Sucht nach Dateien ähnlich [Spotlight](http://www.apfelwiki.de/Main/Spotlight) |
| ls |  | Zeigt den Inhalt des Verzeichnisses an in dem man sich gerade befindet. Besondere Argumente: ls -a zeigt auch versteckte Dateien, ls -l zeigt Zusatzinformationen |
| lsof |  | Zeigt alle geöffneten Dateien an (list open files); z.B. "lsof | grep Finder" für alle vom Finder geöffneten Dateien |
| machine |  | Ausgabe des CPU Typ |
| md5 |  | Überprüfen md5 Prüfsumme |
| md5sum |  | Überprüfen md5 Prüfsumme |
| [man](http://www.apfelwiki.de/Main/Man) <Befehl> |  | Zeigt eine ausführliche Hilfe zum gesuchten Befehl an z.B. man ls |
| mkdir verzeichnis |  | Erstellt ein Verzeichnis verzeichnis. Ist verzeichnis kein absoluter Pfad (mit führendem Slash), wird das neue Verzeichnis im aktuellen Verzeichnis erstellt. |
| mv quelle ziel |  | Verschiebt eine Datei entweder in einen Ordner (wenn ziel der Pfad zu einem Ordner ist) oder benennt sie um. |
| pico <datei> |  | Startet einen kleinen Textbasierten Editor |
| pmset |  | Ausgabe und Setzen von Energiesparparametern |
| pwd |  | Gibt den Pfad des aktuellen Verzeichnisses aus. |
| rm <dateiname> |  | Löscht eine Datei, Datei lässt sich aus Finder direkt ins Terminal ziehen |
| rmdir verzeichnis |  | Löscht ein leeres Verzeichnis- |
| reboot |  | Neustart des Rechners |
| sudo <Befehl> |  | Führt einen Befehl als root aus. Es muss das admin Passwort eingeben werden. Achtung! man sieht bei der Eingabe des Passwortes keine Sternchen oder Ähnliches. |
| [top](http://www.apfelwiki.de/Main/Top) |  | Informationen über laufende Prozesse, Speicherbedarf, CPU Last, Cache |
| vi <datei> |  | Startet einen kleinen Textbasierten Editor |
| [w](http://www.apfelwiki.de/Main/W) |  | Zeigt Übersicht und Aktivität der eingelogten Nutzer |
| which <Terminalprogramm> |  | Zeigt den Pfadnamen zum angegebenen Terminalprogramm |

* **Ordner** mit einem Namen, der aus zwei Wörtern besteht, **öffnen** mit:

VWQ2MTU@VWAGWOLOI0N9 MINGW64 /y

$ cd Eigene\ Dateien/

* **Funktion von Tabulatortaste:** Vervollständigen eines unvollständigen Befehls (wenn der vorhandene Teil eindeutig ist)

z.B. VWQ2MTU@VWAGWOLOI0N9 MINGW64 /y

$ cd Eig

Tabulatortaste → VWQ2MTU@VWAGWOLOI0N9 MINGW64 /y

$ cd Eigene\ Dateien/

**Internetquellen:**

<http://magazin.phlow.de/terminal/>

<http://www.apfelwiki.de/Main/Terminal>

https://wiki.ubuntuusers.de/Shell/Einf%C3%BChrung

**Erste Schritte mit Git**

* Installieren von der Git-Software für das jeweilige Betriebssystem
* Bei Git Hub anmelden
* TortoiseGit aus dem i.do.Store runterladen

Wenn man nach dem Installieren der Git-Software git in die Konsole eingibt:

$ git

usage: git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

These are common Git commands used in various situations:

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

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

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

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

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

bisect Use binary search to find the commit that introduced a bug

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

checkout Switch branches or restore working tree files

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

merge Join two or more development histories together

rebase Forward-port local commits to the updated upstream head

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

1. **Klonen eines repositorys von Git Hub in den Ordner Test auf dem Desktop**

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~

$ cd

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~

$ cd Desktop

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop

$ cd Test

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Test

$ git clone https://github.com/HBorrek/TestRepo.git

1. **Dateien beim „Pushen“ zum Server ignorieren**

Befehl:

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/gitTests

$ git init

Initialized empty Git repository in C:/Users/VWQ2MTU/Desktop/gitTests/.git/

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/gitTests (master)

$ touch .gitignore

* Es wird eine Textdatei .gitignore erstellt. Da kann man alle Dateien hinein schreiben, die ignoriert werden sollen, also nicht zum Server „gepusht“ werden sollen.

10.12.2015

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~

$ cd Desktop

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop

$ cd Probe

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe

$ git init

Initialized empty Git repository in C:/Users/VWQ2MTU/Desktop/Probe/.git/

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git clone ~/Desktop/Probe

Cloning into 'Probe'...

warning: You appear to have cloned an empty repository.

done.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git add Etc.txt

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git commit -m "Commit-Nachricht"

[master (root-commit) 102848a] Commit-Nachricht

1 file changed, 14 insertions(+)

create mode 100644 Etc.txt

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git push origin master

fatal: 'origin' does not appear to be a git repository

fatal: Could not read from remote repository.

Please make sure you have the correct access rights

and the repository exists.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git remote add origin VWQ2MTU

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git checkout -b feature\_x

Switched to a new branch 'feature\_x'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (feature\_x)

$ git checkout master

Switched to branch 'master'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git branch -d feature\_x

Deleted branch feature\_x (was 102848a).

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git checkout -b branch\_one

Switched to a new branch 'branch\_one'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (branch\_one)

$ git push origin branch\_one

fatal: 'VWQ2MTU' does not appear to be a git repository

fatal: Could not read from remote repository.

Please make sure you have the correct access rights

and the repository exists.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (branch\_one)

$ git checkout master

Switched to branch 'master'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git push origin branch\_one

fatal: 'VWQ2MTU' does not appear to be a git repository

fatal: Could not read from remote repository.

Please make sure you have the correct access rights

and the repository exists.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git checkout branch\_one

Switched to branch 'branch\_one'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (branch\_one)

$ git remote add origin VWQ2MTU

fatal: remote origin already exists.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (branch\_one)

$ gut checkout master

bash: gut: command not found

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (branch\_one)

$ git checkout master

Switched to branch 'master'

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git remote add origin VWQ2MTU

fatal: remote origin already exists.

VWQ2MTU@VWAGWOLOI0N9 MINGW64 ~/Desktop/Probe (master)

$ git pull

fatal: 'VWQ2MTU' does not appear to be a git repository

fatal: Could not read from remote repository.

Please make sure you have the correct access rights

and the repository exists.

* **Befehle für Remote Repositorys:**

1. git remote: Zeigt Server an, der für das Projekt lokal konfiguriert ist und listet die

Kurzbezeichnungen für die vorhandenen Remote Repositorys auf

2. git remove –v: Anzeigen der jeweiligen URL für jeden Kurznamen, die Git gespeichert hat.

3. git remote add [shortname] [url]: Hinzufügen eines neuen Repositorys mit einem Kurznamen.

4. git fetch [remote-name]: Herunterladen von Daten aus Remote Repository

-> nach dem Herunterladen kennt das eigene Repository Verweise auf alle Branches in dem Remote Repository, die man mit eigenen Branches zusammenführen oder durchschauen kann.

5. git push [remote-name] [branch-name]: Hochladen der Änderungen in ein gemeinsames

Repository.

6. git remote show [remote-name]: Remote Repository nach Informationen durchstöbern.

7. git remote rename [alter Name] [neuer Name]: Referenz auf ein Remote Repository

umbenennen.

8. git remote rm [Name von Remote Repository]: Referenz entfernen.

1. **Pushen**
2. Schritt: in Git Hub neues repository erstellen (hier: borrekRepo)
3. Ordner für das repository erstellen
4. In Konsole leeres repository erstellen
5. Textdokumente erstellen und zum repository hinzufügen
6. Remote repository erstellen und pushen

Terminal:

Darshans-iMac:~ heideroseborrek$ cd\

Darshans-iMac:~ heideroseborrek$ cd Desktop\

Darshans-iMac:Desktop heideroseborrek$ mkdir borrekRepo\

Darshans-iMac:Desktop heideroseborrek$ cd borrekRepo\

Darshans-iMac:borrekRepo heideroseborrek$ git init\

Initialized empty Git repository in /Users/heideroseborrek/Desktop/borrekRepo/.git/\

Darshans-iMac:borrekRepo heideroseborrek$ dir README.rdoc\

-bash: dir: command not found\

Darshans-iMac:borrekRepo heideroseborrek$ touch README.rdoc\

Darshans-iMac:borrekRepo heideroseborrek$ git status\

On branch master\

\

Initial commit\

\

Untracked files:\

(use "git add <file>..." to include in what will be committed)\

\

\cf2 README.rdoc\cf0 \

\cf2 textREADME.rtf\cf0 \

\

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

Darshans-iMac:borrekRepo heideroseborrek$ git .add\

git: '.add' is not a git command. See 'git --help'.\

\

Did you mean this?\

add\

Darshans-iMac:borrekRepo heideroseborrek$ git add --all\

Darshans-iMac:borrekRepo heideroseborrek$ git commit -m "first commit"\

[master (root-commit) e73e8f8] first commit\

Committer: Heiderose Borrek <heideroseborrek@Darshans-iMac.local>\

Your name and email address were configured automatically based\

on your username and hostname. Please check that they are accurate.\

You can suppress this message by setting them explicitly:\

\

git config --global user.name "Your Name"\

git config --global user.email you@example.com\

\

After doing this, you may fix the identity used for this commit with:\

\

git commit --amend --reset-author\

\

3 files changed, 14 insertions(+)\

create mode 100644 README.rdoc\

create mode 100644 sachen.rtf\

create mode 100644 textREADME.rtf\

Darshans-iMac:borrekRepo heideroseborrek$ git remote add origin https://github.com/HBorrek/borrekRepo.git\

Darshans-iMac:borrekRepo heideroseborrek$ git push -u origin master\

Counting objects: 5, done.\

Delta compression using up to 8 threads.\

Compressing objects: 100% (4/4), done.\

Writing objects: 100% (5/5), 601 bytes | 0 bytes/s, done.\

Total 5 (delta 1), reused 0 (delta 0)\

To https://github.com/HBorrek/borrekRepo.git\

\* [new branch] master -> master\

Branch master set up to track remote branch master from origin.

1. **Branches erstellen und verwalten**
2. Einen neuen Branch lokal erstellen und „öffnen“

$ git checkout -b [name\_of\_your\_new\_branch]

1. Den Branch nach extern (GitHub) pushen:

$ git push origin [name\_of\_your\_new\_branch]

1. Sich alle vorhandenen Branches anzeigen lassen:

$ git branch

Beachte: wenn man etwas in seinem Branch commiten möchte sollte man sicher gehen, dass man sich im richtigen Verzeichnis/ im richtigen Branch befindet!

1. Eine neue Remote Version für den Branch erstellen:

$ git remote add [name\_of\_your\_remote]

1. Veränderungen vom Commit in den Branch pushen:

$ git push origin [name\_of\_your\_remote]

1. Branch aktualisieren, wenn der Original-Branch vom Original-Repository verändert wurde:

$ git fetch [name\_of\_your\_remote]

1. Änderungen zusammenführen, z. B. wenn der Branch abgeleitet ist aus dem develop-Branch:

$ git merge [name\_of\_your\_remote]/develop

1. Einen Branch lokal löschen:

$ git branch -d [name\_of\_your\_new\_branch]

1. Verstärktes Löschen von dem Branch im lokalen Dateisystem:

$ git branch -D [name\_of\_your\_new\_branch]

1. Den Branch von GitHub löschen:

$ git push origin :[name\_of\_your\_new\_branch]

**Internetquellen:**

<http://guides.railsgirls.com/github/>

<https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches>

<https://www.ralfebert.de/git/branches-mergen/>

http://stackoverflow.com/questions/25649374/git-merge-local-with-remote-master-using-terminal