**Anleitung : STU3 : Git Hub for Windows**

* *Wer sich STU 3 auf seinem Rechner installieren will, wird enttäuscht sein. STU3 Beta wurde mit mehreren spezielien (Repository) PROGRAMMTEILEN über Github direkt konfiguriert. Die exakten Pfadangaben aus dem Original muss per Hand angepasst werden, was im Normalfall sehr viel Programmierkenntnisse voraussetzt. Ohne die exakten Pfadangaben aus dem Original wird STU 3 auf eurem Rechner nicht laufen.*
* *Es gibt zur Steuerung von GITHUB auch grafische Oberflächen, die die folgenden Grundkenntnisse ebenfalls benötigen. Sie werden daher nicht empfohlen.*
* *An manchen Stellen wird nach der Ausführung der hier beschriebenen Befehle nichts passieren am Bildschirm. Das ist normal. Daher wird immer wieder mit Hilfsbefehlen der Stand der Updates überprüft.*
* *Die hier aufgeführten Arbeitsschritte sind täglich immer wieder notwendig und pro einzelnes Repository. Es ist wichtig, dass nur in der richtigen Reihenfolge auch Änderungen problemlos funktionieren. Wenn es Probleme gibt, so bitte an an das Team wenden. ggf. kann im Einzelfall eine Teamviewer-Session oder Absprachen im hauseigenen DISCORD Channel in den Fall weiterhelfen*

***Als wichtiges Arbeitsvideo empfehlen wir hier, das von Huxinator erstellte Arbeitsvideo zum Bearbeiten von github. Hierbei wird allerdings NICHT die Ersteinrichtung beschrieben.***

**Legende:**

* GIT BASH = von git 4 windows. muss installiert sein und wird in der Anleitung GRÜN angezeigt → der Hauptbefehl bei allen Eingaben ist immer “git”...
* BROWSER = Dein eigener Internetbrowser , der im Fenster [www.github.com](http://www.github.com) anzeigt, wird in dieser Beschreibung BRAUN angezeigt.
* VERZEICHNIS AM RECHNER = Dein vom Internet kopierter 1:1 auf Stand gebrachtes Programm für STU 3 wird hier ORANGE angezeigt
* BESCHREIBUNG VON SPEZIAL-AUSDRÜCKEN = werden VIOLETT angezeigt
* Wichtige ANMERKUNGEN = werden in ROT beschrieben

GRUNDLAGEN:

a) **Einen Github Account anlegen**

--><https://www.github.com> → registrieren, anmelden, einloggen.

Benutzername und Passwort : nach Wahl. Email: kann fiktiv sein, sollte echte Email sein, denn über diese Email werden alle Veränderungen an abonnierten Dateien mitgeteilt

b)  **git for windows auf dem Rechner installieren.**alles mit Standart-Einstellungen → <https://gitforwindows.org/>

MEINE EIGENE STU3 PROGRAMMIERSTELLE ERÖFFNEN

c) **Forken***(Aufgabeln)*   
  
--> Ins Github auf dem BROWSER und auf <https://www.github.com> gehen. Um einen Fork zu erzeugen, klickt man im Internet bei github.com einfach auf den Fork-Button auf der entsprechende Seite (siehe untenstehende Links)  
  
bisherige Repos für STU 3 sind:

<https://github.com/usox/stu3> → Stu 3 Programmierdaten

<https://github.com/usox/stu3-assets> → Stu 3 Bilder

<https://github.com/usox/stu3-doku> → STU 3 Python 2 Dokumentation (wiki)

(weitere in Vorbereitung…)

* Dort die notwendigen Dateien "aufgabeln" = "Forken" . Das Aufgabeln = "Forken" eines Repositories(\*) stellt eine 1:1 Kopie eines bestehenden Repositories her. Jeder Benutzer hat seinen eigenen Fork und arbeitet nur damit (und auch nur mit einem eigenen Arbeitsbereich). Hier wird das Einrichten einer “Master” Datei erklärt.

\* (Repository = Das ist eine Art von Ablage von Files und Daten im Internet, die notwendig sind, um mit Github am eigenen Rechner mitzuarbeiten. Das kann man sich in etwa wie bei Google Drive ein "gezippedes Verzeichnis im Internet” vorstellen, aber statt zip bekonnt das ganze die Endung ".git" und am heimischen Rechner liegt es im Klartext entzipped vor, um damit arbeiten zu können)

d) **GIT FOR WINDOWS starten**   
also: GIT BASH starten. (Ab hier arbeiten wir bis auf Widerruf mit der GIT BASH)

e) **Clone**   
  
(Kopieren aller Daten eines Repository auf den eigenen Rechner)

* Damit man die Daten des geforkten Repositories auf den Rechner bekommt, muessen die Daten auf den Rechner gezogen (also kopiert) werden. Dies nennt man `clonen`. Die Url fuer den Fork findet man auf der Github-Seite des neu erstellen Forks   
  (\* deingithub => der Seite unter der die aufgegabelten / geforkten Dateien angezeigt werden)

→ git clone [https://github.com/deingithub(\*)/stu3](https://github.com/deingithub(*)/stu3)

→ git clone [https://github.com/deingithub(\*)/stu3-assets](https://github.com/deingithub(*)/stu3-assets)

→ git clone [https://github.com/deingithub(\*)/stu3-doku](https://github.com/deingithub(*)/stu3-doku) usw.  
Wichtig: Bei der Installation von "git for windows" wird standartisiert IMMER das %home Verzeichnis im Windows genutzt. Das ist üblicherweise "C:\Users\des Benutzers eigene Dateien".

* Um dorthin zu wechseln, in der "git bash" den Befehl :

→ cd stu3-assets oder → cd stu3 oder → stu3-doku usw. eingeben

ICH VERBINDE MICH ERSTMALS MIT STU3

f) **Upstream Remote**   
( *= Wir stellen eine grundsätzliche Verbindung zu den Originaldateien von STU3 her)*

* Um eine Verbindung zum original repository herstellen zu koennen, sollte man dieses als neues sog `remote` hinzufuegen. Ein standardisierter Name dafuer lautet `upstream`. Dabei wird erstmals klar, dass es sich bei den Github-Dateien um komprimierte Dateien handelt, da wir hier ausnahmsweise einmalig die Endung ".git" benutzen.

→ git remote add upstream https://github.com/st-universe/assets.git

→ git remote add upstream <https://github.com/st-universe/docu.git>

→ git remote add upstream <https://github.com/st-universe/starsystem-generator.git>

→ git remote add upstream <https://github.com/st-universe/planet-generator.git>

→ git remote add upstream <https://github.com/st-universe/planet-generator.git>

→ git remote add upstream <https://github.com/st-universe/forum-skin>

* Um festzustellen ob dabei überhaupt irgendetwas passiert ist, empfehlen wir den folgenden Befehl einzugeben, der vergleicht, ob alles geklappt hat.

→ git remote -v  
*(-v verbose = Überprüfe die Remoteverbindung auf Fehler)*

(zum Beispiel sollte die Verbindung dann in etwa so aussehen...

***$ git remote -v***

***origin https://github.com/deingit/stu3-assets (fetch)***

***origin https://github.com/deingit/stu3-assets (push)***

***upstream https://github.com/st-universe/assets.git (fetch)***

***upstream https://github.com/st-universe/assets.git (push)***

* Mit dem nächsten Befehl überprüfen wir zur Sicherheit ob unsere geforken Original-Master-

Dateien noch auf Stand mit unserem Originalmaster von STU 3 sind. ("aktualisiere die Verbindung").

→ git fetch upstream

* MIt dem folgenden Befehl wird gebeten, die ganzen Updates wenn möglich auf aktuellem Stand zu bringen, das ist eine notwendige Sicherheitsmaßnahme, die hier nicht näher beschrieben wird.

→ git rebase upstream/master

* Wir vergewissern uns nochmals, was wir gerade getan haben, indem wir den "Status überprüfen" mit:

→ git status  
(Hierbei könnten uns am Bildschirm ROT markierte evtl. Änderungen der Dateien vom Original entgegenspringen)

ICH ARBEITE ERSTMALS MIT STU3,  
ÄNDERE DABEI DATEIEN UND TEILE DIESE RICHTIG MIT

g) **Branch eröffnen**   
*("Eine Abteilung" → entspricht dem eigentlichen Arbeitsplatz)*Jedes git-Repository hat eine Haupt-Branch, genannt `master`.   
**Aber es wird AUSSER FÜR UPDATES NIEMALS!!! mit der master-Branch gearbeitet.**

* *Branches stellen eigene Arbeitsbereiche oder "Abteilungen" dar*. Der Vorteil dabei ist, daß man nichts an den "aufgegabelten" also "geforkten" "master-Dateien" verändert. Man kann beliebig viele Branches/Arbeitsbereiche erstellen und soll das auch tun. Änderungen, die in einer Branch *commited (mitgeteilt)* werden, sind auch NUR in dieser *Branch (Abteilung*) enthalten. Man erzeugt für jede einzelne Änderung auch gen. “Ticket oder PullRequest” immer wieder eine neue Branch.

→ git checkout -b NeueBranch

*(im Klartext: git eröffne -b → Build die Abteilung NeueBranch)*

* Nun befindet man sich auch schon in dem neueröffneten Arbeitsbereich. *Mit dem Befehl git checkout <BRANCHNAME>` kann man nun zwischen den Braches (Arbeitsbereichen) umherwechseln.*

→ git checkout master

wechselt z.B. immer zur (Hauptabteilung unseres geforkden Masters   
VORSICHT!!!! NICHT MASTER BRANCH BEARBEITEN

→ git checkout NeueBranch wechselt zurueck in unseren gerade angelegten, neuen Branch.

h) **Aenderungen vornehme**n   
*(an den am Rechner vorhandenen Dateien in den Windows Verzeichnissen in $home)*

* Nun koennen beliebige Aenderungen an den Dateien vorgenommen werden. Um bei unseren Beispiel zu bleiben, legen wir ein neues Bild für die stu 3-assets (STU 3 Grafik-Erweiterungen) an und stellen das Bild in einem Ordner am Rechner der normalerweise “fields” heisst, Das Abbild der STU 3 Dateien liegt normalerweise im $home Ordner, der bei normalen PCs hier zu finden ist:

→ "C:\Users\des Benutzers eigene Dateien"

→ “C:\Users\des Benutzers eigene Dateien\stu3-assets\fields”

* Wir ändern unsere Datein und wir wechseln erst dann wieder in die Bash und überprüfen die Änderungen mit :

→ git status

Der Befehl zeigt danach alle geänderte Dateien in ROT   
und neue Dateien in GRÜN oder WEISS an

h) **Datei(en) adden**   
*( = diese Änderungen Github vorab theoretisch mitteilen).*

* Notwendig um github Änderungen von unserem Rechner weg ins Internet vorab bekannt zu machen. Das folgende Beispiel würde alle Bilder in dem Ordner "fields" in unserem $home Verzeichnis am Windows Rechner aktualisieren, das dient nur dafür, dass die neuen Dateien auch irgendwo bekannt werden )

→ git add fields

*(Es wird hier ausdrücklich NICHT empfohlen einzelne Dateien zu adden oder zu commiten, da sämtliche Änderungen innerhalb eines Ordners von git automatisch erfasst werden)*

i). **Commit erstellen**   
*(eine kurze Beschreibung für die neuen geänderten Dateien mitteilen )  
(Wichtig, das wird gern vergessen)*

* Ein Commit umfasst alle Aenderungen, die man in einem Arbeitsschritt vornehmen moechte. Der Commit wird dabei nur in der gerade aktiven Branch angelegt. Es reicht diesen HIER kurz zu fassen, da der Commit später im Github im Internet selbst nochmal genauer beschrieben werden kann. (*Hier nur am Rande erwähnt: Der Commit wird ueber eine eine eindeutige ID referenziert.)*

→ git commit -m NeueBranch

(-m = Merge, hinzufügen , also eigentlich: git übertrage meine folgende Kurzbeschreibung und füge diese den Original STU3 Dateien von USOX aus meiner “NeuenBranch”hinzu.)

* Wir überprüfen auch hier wieder unsere Änderungen mithilfe des folgenden Befehls.
* Nach der Eingabe `git log` sollte nun der Commit an erster Stelle erscheinen. Alle anderen Commits beschreiben SÄMTLICHE Änderungen von oben nach unten chonologisch, die seit dem Eröffnen der Programmierstelle geändert wurden - das kann seeehr lang lang)

-----> git log

j). **Branch pushen**   
*(Den Inhalt des geänderten Arbeitsplatzes endgültig mitteilen,   
den Arbeitsplatz wieder saubermachen, Die Abteilung danach schließen)*

* Github überträgt mit dem folgenden Befehl erst alle geänderten Daten tatsächlich an die Master-Dateien von usox/stu3… - das nennt man `pushen`.

----> git push origin NeueBranch

(im Klartext: git übertrage/pushe jetzt endgültig die geänderten Dateien aus “NeueBranch” zum absoluten Ursprungsort (usox/stu3)

* Nun wurde die Branch (Der von erstellten Arbeitsabschnitt oder Bereich) zu github (zum Origin(al)-Master) uebertragen.

**k) Pull Reques**t !

( = Antrag auf letztendliche Änderung der Originaldateien von STU 3)  
Ein PullRequest ist ein "Anfrage" an das original master repository, die Aenderungen, die man selbst in seiner eigenen Branch gemacht hat durchzuführen.

* **Der Pull Request funktioniert anders als die ganzen bisherigen Arbeitsschritte.**

**über den BROWSER IM INTERNET direkt im** [**https://www.github.com**](https://www.github.com) **. Sind die Arbeitsschritte bisher richtig durchgeführt so kommt mit grünem Kästchen am Bildschirm “Compare & Pull Request” → Dort draufdrücken.**

* **Der Pull Request selbst wird durchgeführt in eurem eigenen “eröffneten Branch"**
* **ABER: NIEMALS im geforkten master Repository**
* Man wechselt zur “Branch”” seiner eigenen Daten und öffnet einen Pull Request. Darin beschreibt man nochmal die ganzen Änderungen AUSFÜHRLICHST im Klartext, die man schon bereit haben sollte.
* Dann wird der Pull Request an den echten Master der STU Dateien geschickt, mit der Bitte alle diese Änderungen ins Programm einzupflegen.
* Sobald der Pull-Request erstellt worden ist, wird der Antrag auf Änderung von usox/stu3 entschieden. Wird er angenommen, so werden sämtliche Änderungen am Programmcode ins aktuelle STU3 eingefügt
* wird der Antrag abgelehnt, so gibt es logischerweise auch keinerlei Änderungen.

l) Fork synchronisieren  
**(Wir schreiben die Änderungen auf unseren geforkden Master, auch: tägliche Aktualisierung)**

* Wenn der Antrag Erfolg hatte, dann ist es mit den geänderten Dateien noch nicht getan.Denn nun muss man dafür Sorge tragen, daß auch die “geforkeden master””Dateien aktualisiert sind. Dazu müssen auch diese Fork-Daten synchronisiert werden.
* **Dieselben Arbeitsschritte sind TÄGLICH durchzuführen, um auf dem Stand zu bleiben, nur hier und nur hier arbeiten wir an den “geforkden Master-Dateien”**
* Dazu wechselt mal zünächst in die master-branch:

→ git checkout master

* Man holt sich die Aenderungen des Upstream-Remote (s.3.)

→ git fetch upstream

* jetzt fuegt man die Daten mit seiner eigenen “Branch” zusammen.

→ git merge upstream/master  
  
Nun sollte man wieder auf dem aktuellen Stand sein.

**ENDE**

*Als weiterführende und übersichtliche Aufstellung in Kurzform der täglichen Arbeitsschritte wird die folgenden Datei “GITHUB 4 WINDOWS - LAUFENDES ARBEITEN” empfohlen.*