Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

Grundlagen Rechnernetze und Verteilte Systeme

IN0010, SoSe 2016

Assignment 1 – Git, VM & FizzBuzz (1 Punkt)

Abgabe bis spätestens 01.05.2016, 23:59 MESZ über Git.

Übersicht

VServer und Versionsverwaltung (GIT)

Im Rahmen der Programmieraufgaben stellen wir jedem zur Vorlesung angemeldeten Studierenden jeweils einen virtuellen „Rootserver / VServer“ (VM) mit öffentlicher IPv4-Adresse und eigenem IPv6-Subnetz zur Verfügung. Die VMs dienen als einheitliche Testplattform, auf denen Ihre kompilierbar und ausführbar sein müssen. Der Zugang zu den VMs erfolgt ausschließlich über SSH.

Da die Bereitstellung von mehr als 800 VMs eine beträchtliche Anzahl echter Hardware zur Virtualisierung erfordert, haben wir uns zu folgenden beiden Schritten entschieden:

- VMs müssen einmalig angefordert werden.
- Inaktive VServer werden nach 25 h abgeschaltet (PowerOff[1]) und müssen neu gestartet werden.

Die Anforderung bzw. das Wiedereinschalten von VServern geschieht vollautomatisch und innerhalb weniger Minuten.

Die Abgabe der Programmieraufgaben geschieht über Git. Dabei handelt es sich um eines der gängigen Versionsverwaltungssysteme, welche in der Softwareentwicklung eingesetzt werden. Es ermöglicht mehreren Programmierern gemeinsam an einem Softwareprojekt zu arbeiten. Quelltexte werden in einem **Repository** abgelegt, welches auf einem Server bereitliegt. Von dort kann der Inhalt des Repositories geklont werden, so dass eine lokale Kopie vorliegt. Auf der Kopie kann nun gearbeitet werden. In regelmäßigen Abständen sollte ein **commit** durchgeführt werden, welcher Änderungen in der lokalen Kopie des Repositories abspeichert. Dabei werden im Repository lediglich die Änderungen gegenüber der letzten Revision gespeichert. Um diese Änderungen auf den Server zu übertragen, muss ein **push** ausgeführt werden. Andere Benutzer, wie z.B. die Übungsleitung, können diese Änderungen mittels eines **pull** herunterladen und so ihre lokale Kopie aktualisieren. Ausführliche Informationen über Git finden Sie in [1].

Warum brauchen wir Git?

Es gibt mehrere Gründe, weswegen wir für die Programmieraufgaben Git verwenden:

1. Jeder Informatiker sollte mit wenigstens einem Versionsverwaltungssystem umgehen können. Falls Sie es nicht schon längst können, sollten Sie es besser früher als später lernen.
2. Wir brauchen ein System, mit welchem die Programmieraufgaben abgeben werden können. Anstatt uns Ihre Porgramme via Email zu schicken oder als Tarball über ein Webformular hochzuladen, müssen Sie nur sicherstellen, dass Ihre aktuelle Version im Git liegt.
3. Es ermöglicht uns, Ihnen auf elegante Art die Vorlesungsunterlagen bereitzustellen: Sie müssen lediglich die loakle Kopie Ihres material-Gits aktualisieren (ein Befehl oder zwei Klicks) und nicht jeden Downloadlink einzeln anklicken. Außerdem sehen Sie auf einen Blick, was sich wann geändert hat.

[1] Es gibt keine Snapshots, d. h., offene nicht gespeicherte Daten gehen verloren.

Prof. Dr.-Ing. Georg Carle
carle@tum.de
Johannes Naab, Stephan Günther, Maurice Leclaire, Markus Ongyerth, Lukas Erlacher
grnvs@net.in.tum.de

Wo bekomme ich Git?

- Linux-Nutzer installieren `git` mit der Paketverwaltung ihrer Wahl.
- Nutzer von OS X erhalten `git` entweder über XCode Kommandozeile-Tools [2] oder über MacPorts bzw. Brew.
- Windows-Nutzer wollen sicherlich einen grafischen Client. Empfehlenswert ist hier `Git for Windows` [3], welcher sich direkt in das Kontextmenü des Windows-Explorers integriert.

SSH Schlüsselpaar generieren

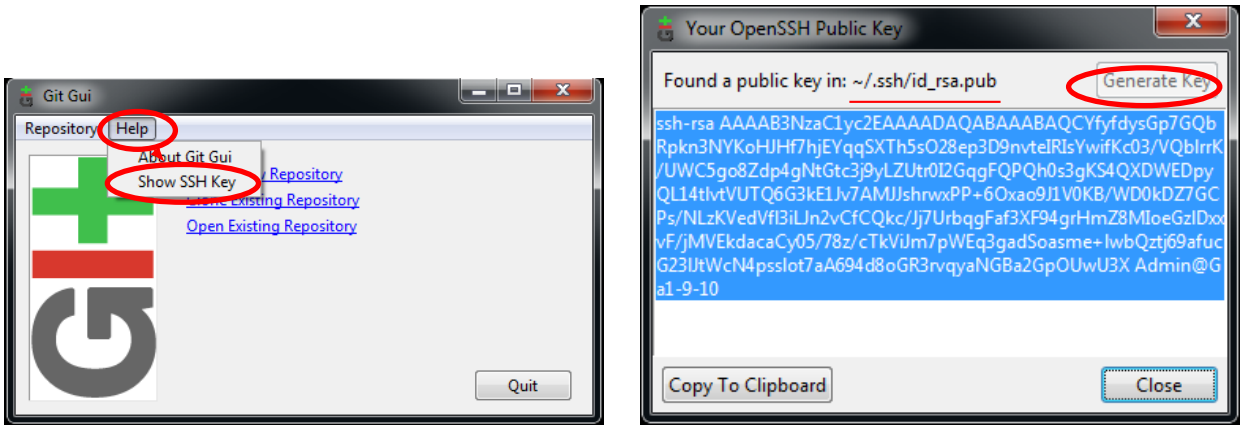
Um auf die VM und Git Zugriff zu bekommen, müessen Sie ein Schlüsselpaar generieren, welches für den Zugriff per SSH geeignet ist. In unserem Beispiel verwenden wir RSA mit einer Länge von 2048 bit. Dies garantiert Kompatibilität. Das System des Lehrstuhls unterstützt auch die Schlüsseltypen ED25519 und ECDSA.

Kommandozeile Mit diesem Befehl erzeugen Sie ein RSA-Schlüsselpaar:

```
~ $ ssh-keygen -t rsa -b 2048
```

Sie werden gefragt, wo der neue SSH-Schlüssel gespeichert werden soll. Standardmäßig wird der private Schlüssel als Datei nach `~/.ssh/id_rsa` geschrieben. Der öffentliche Schlüssel wird unter `~/.ssh/id_rsa.pub` abgelegt. Sofern Sie nicht bereits einen SSH-Schlüssel verwenden, brauchen Sie hier nichts zu ändern. Wenn Sie den Befehl ausführen, wird Sie das Programm nach einem Passwort für den neuen privaten Schlüssel und Informationen über Sie fragen. Die Informationen müssen Sie nicht ausfüllen.

GUI Auf Windows kann es einfacher sein, das Schlüsselpaar mit der GUI zu generieren. Gehen Sie dazu wie in Abbildung 1 gezeigt vor.



(a) **(b)**

Abbildung 1: (a) Klicken Sie auf `Help` und dann auf `Show SSH Key`. Dann öffnet sich das Fenster in Abbildung 1b. (b) Falls noch kein Schlüsselpaar vorhanden ist, können Sie einen neuen Schlüssel mit `Generate Key` generieren. Wenn schon ein Schlüsselpar vorhanden ist, können Sie hier den Pfad sehen (den werden Sie für SSH später brauchen) und den öffentlichen Schlüssel bequem kopieren.

Nun können Sie auf Moodle den öffentlichen Schlüssel (`<pfad>.pub`) hochladen und abgeben. Der private Schlüssel wird Ihnen dann den Zugriff auf Ihr persönliches Git sowie das material-Git ermöglichen.

Git Repository Klonen

Zunächst müssen Sie Ihr Arbeitsverzeichnis für die Programmieraufgaben auschecken:

Den Namen des Repositorys kann man mit ssh erfahren:

```
~ $ ssh grnvs@git.net.in.tum.de
```

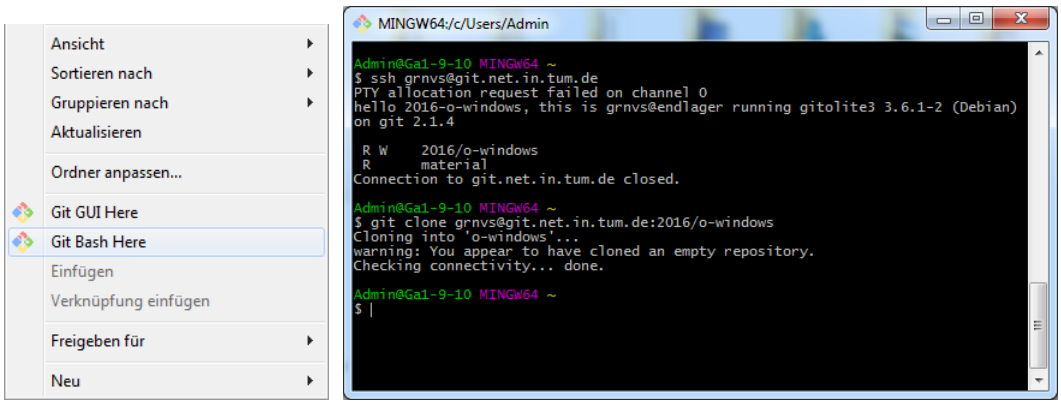
Kommandozeile Der folgende Befehl klont das Git Respository:

```
~ $ git clone grnvs@git.net.in.tum.de:2016/<Matrikelnummer> <Zielverzeichnis>
```

Die Host Key Fingerprints für `git.net.in.tum.de` lauten:

```
2048 SHA256:BsRgpGG9GucsvJLb3QQBIXTwJu5PBRIBVraS7ZORZ8M git.net.in.tum.de (RSA)
256 SHA256:T1+LOSUSN6sw5G/oKZnw+330k45/aLLk0ylS6bxo8Wc git.net.in.tum.de (ECDSA)
256 SHA256:wv2E66tEog3wVDZ6kGYN4OWNuLb5RQvpBxn5BLBU2Mg git.net.in.tum.de (ED25519)
2048 MD5:63:d5:c9:13:2d:ff:f4:22:e8:ea:f6:3d:dc:1d:e4:cc git.net.in.tum.de (RSA)
256 MD5:99:95:a3:e9:2a:05:a5:d3:c5:5c:47:54:8d:15:1e:a8 git.net.in.tum.de (ECDSA)
256 MD5:d8:71:9f:6f:3f:7a:8e:87:06:d6:8f:23:07:f1:5f:7a git.net.in.tum.de (ED25519)
```

Git for Windows Folgen Sie der Anleitung in Abbildung 2. Anschließend sollten Sie ein leeres Git Repository vorfinden. Für die einzelnen Programmieraufgaben müssen zu gegebener Zeit die Unterverzeichnisse `assignmentX` erzeugt werden, innerhalb welche die jeweilige Programmieraufgaben bearbeitet wird.



(a) (b)

Abbildung 2: (a) Rechtsklick im Explorer, dann "Git Bash". (b) Den Repositorynamen mit ssh herausfinden und per cli klonen.

Hilfe zu Git Für die Verwendung von Git wird auf folgende externe Tutorials verwiesen:

- `https://rogerdudler.github.io/git-guide/index.html`
- `https://try.github.io/`
- `https://git-scm.com/docs/gittutorial`

Wie komme ich auf meine VM?

VM anfordern

Wie eingangs erwähnt müssen Sie Ihre VM zunächst anfordern (bzw. einschalten). Dies geschieht dadurch, dass Sie sich per ssh auf die spezielle VM `svm000.net.in.tum.de` verbinden. Diese wird dann automatisch ihre VM für Sie anlegen und starten.

```
~ $ ssh grnvs@svm000.net.in.tum.de
```

Dieser SSH-Dienst gibt den Hostname der erzeugten VM und die Fingerprints der Hostkeys zurück.

Der Host `svm000.net.in.tum.de` ist über folgende Host Key Fingerprints zu identifizieren:

```
2048 SHA256:cfVIkE6jBfT8r+a0LD5r9fqiqX1m4mpEyy1/IkC+dDc svm000.net.in.tum.de (RSA)
256 SHA256:MXmC9zqzMzGxxslgS6eIHDMd6u/x7WFk7glt+i5cbAA svm000.net.in.tum.de (ECDSA)
256 SHA256:1UikPi02XGCFDQPvtUTEgNpyATZ3zICQqCXAbIlXm4w svm000.net.in.tum.de (ED25519)
2048 MD5:bc:26:c9:28:c5:aa:aa:cc:ff:01:b0:f4:93:65:15:bd svm000.net.in.tum.de (RSA)
256 MD5:fa:22:9d:fb:b1:48:8a:31:e0:c4:12:59:14:1d:4c:a0 svm000.net.in.tum.de (ECDSA)
256 MD5:61:0c:91:47:07:06:89:8e:02:be:98:d9:0d:e9:75:16 svm000.net.in.tum.de (ED25519)
```

Login auf der VM

Der Benutzername für die VMs ist `root`. Passwort benötigen Sie keines, die Authentisierung erfolgt über den im Moodle abgegeben SSH-Key.

Kommandozeile Wie üblich:

```
~ $ ssh root@svm<xzy>.net.in.tum.de
```

Windows + MobaXterm Der wahrscheinlich mit Abstand angenehmste und übersichtlichste Weg ist, *MobaXterm* [4] zu verwenden. Dabei handelt es sich um einen SSH-Client mit Unterstützung für SSHFS, integriertem Dateibrowser und einem XServer (den wir nicht brauchen). Es ist kostenfrei erhältlich und es gibt angenehmer Weise sogar eine Standalone-Binary (keine Installation notwendig). Nach der Installation folgen Sie bitte den Hinweisen in Abbildung 3a.

Arbeiten auf der VM

Da vermutlich nicht jeder mit den kommandozeilen-basierten Texteditoren wie `vim` oder `nano` umgehen kann oder will, kann im Anschluss das Homeverzeichnis der VM lokal gemountet werden:

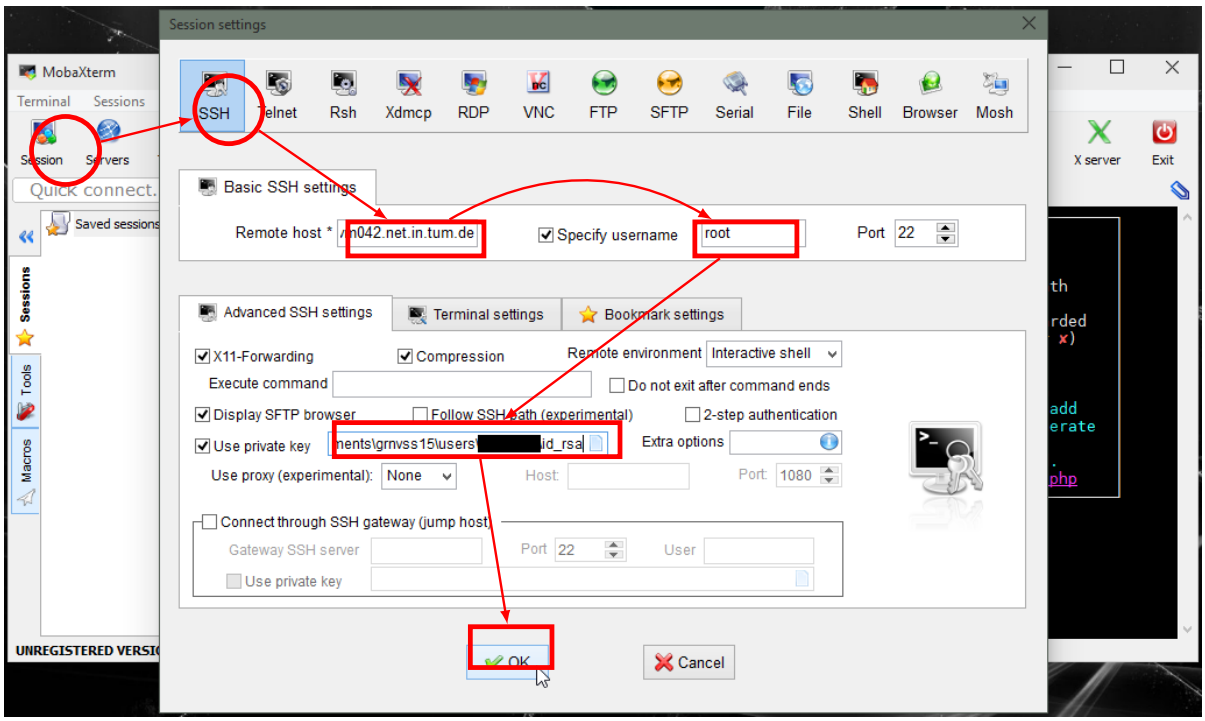
- Linux-Nutzer installieren sich SSHFS über ihren bevorzugten Paketmanager.
- OS X-Nutzer haben es leider etwas schieriger. Aber auch hier gibt es SSHFS-Clients. Eine sinnvolle Anleitung finden Sie unter [5]
- Unter Windows bringt MobaXterm bereits einen SSHFS-Client mitbringt.

Kommandozeile Unter Linux

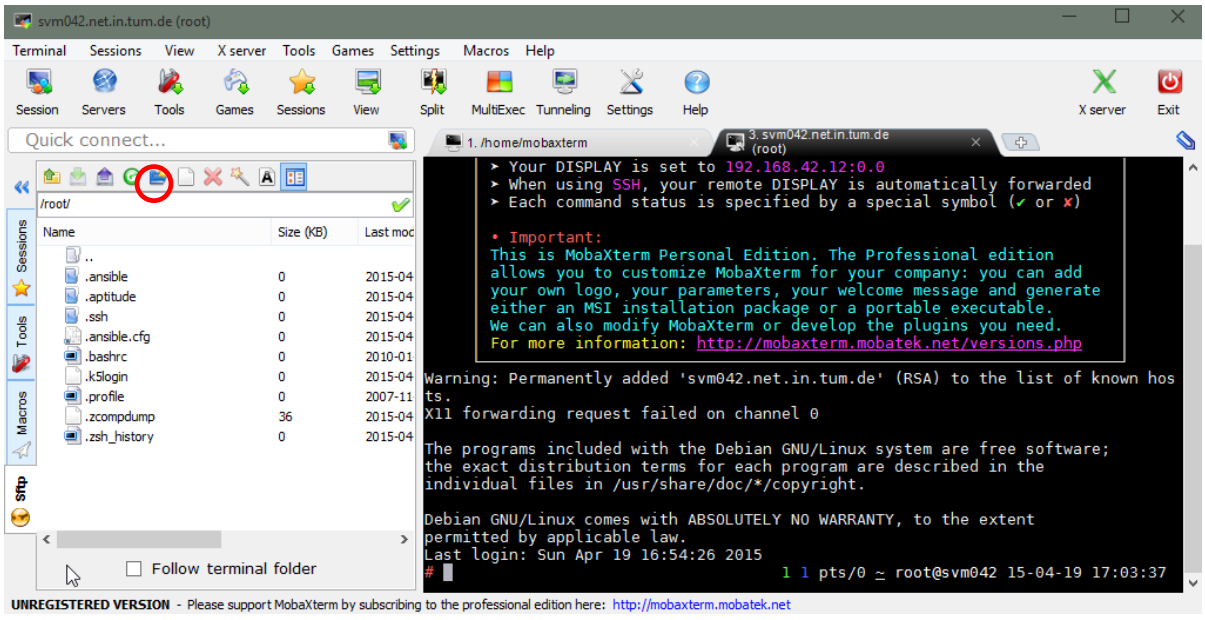
```
~ $ sshfs -i <pfad> root@svm<xzy>.net.in.tum.de: -o allow_other,reconnect
```

und unter OS X

```
~ $ sshfs root@svm<xzy>.net.in.tum.de: -o IdentityFile=<pfad>
```



(a)



(b)

Abbildung 3: (a) Tragen Sie als Remote Host den Namen Ihrer VM ein. Der Benutzername lautet `root`. Geben Sie als „Private Key“den privaten RSA-Schlüssel an (Pfad zur Datei `id_rsa`). Siehe Abbildung 1. Nachdem Sie auf „Ok“ geklickt haben, sehen Sie in der linken Hälfte des Fensters die eben angelegte Session. Ein Doppelklick darauf stellt die Verbindung zur VM her.
(b) Im rechten Teil des Fensters sehen Sie nun die Konsolensitzung. Im linken Teil des Fensters sehen Sie einen Dateibrowser, der Ihnen Zugriff auf das Dateisystem Ihrer VM ermöglicht.
Hinweis: Wenn Sie Dateien auf der Konsole angelegt oder gelöscht haben, müssen Sie unter Umständen Die Dateiansicht des Browsers aktualisieren. Danke roter Kreis.

Alternative können Sie natürlich auch auf einer lokalen Kopie des Git-Repositories arbeiten. Allerdings müssen Sie es dann jedes mal, wenn Sie es auf der VM testen wollen, zuerst committen und pushen und eine zweite Kopie des Repositories auf der VM updaten [6, 7]. Selbstvertsändlich können Sie auch Ihren eigenen Rechner zur Entwicklung verwenden. Allerdings müssen Sie sicher stellen, dass die Abgabe am Ende auf den VMs lauffähig ist. Außerdem benötigen Sie Linux. Unter OS X und FreeBSD unterscheiden sich zumindest Name und Pfad einiger Header. Unter Windows wären die Änderungen deutlich umfangreicher, weswegen wir dringend davon abraten.

Wie gehts jetzt weiter?

Machen Sie sich bitte mit Git und Ihrer VM vertraut. Versuchen Sie, Dateien auf die VM zu übertragen und ein einfaches „TUMGRNVS“ in einer Sprache Ihrer Wahl zu schreiben. Bei Problemen besuchen Sie bitte eine der Programmieruebungen:

Gruppe	Wochentag	Uhrzeit	Raum	Tutor
Mo-P1	Montag	12:00 – 14:00	00.13.054	Lukas Erlacher
Di-P1	Dienstag	12:00 – 14:00	03.07.023	Lukas Erlacher
Mi-P1	Mittwoch	12:00 – 14:00	03.07.023	Markus Ongyerth
Do-P1	Donnerstag	12:00 – 14:00	03.07.023	Markus Ongyerth

Tabelle 1: Programmiergruppen

Bitte beachten Sie zu den Programmierübungen folgende Regeln:

1. Die Programmierübungen beginnen am Dienstag, den 19. April, und finden fortan **nur dann statt, wenn gerade eine Programmieraufgabe zu bearbeiten ist**.
2. Bitte kommen Sie nur mit konkreten Fragen oder Problemen zu den Programmiertutoren – sie sind kein First-Level-Support bei Computerproblemen.

Akademisches Fehlverhalten, Plagiate, Gruppenarbeit und StackOverflow

Die Abgaben werden sowohl automatisch als auch manuell auf Plagiate überprüft. Sollten sich hierbei eindeutige Hinweise auf Übereinstimmen gefunden werden, so werden die betroffenen Teilnehmer vollständig aus dem Bonusverfahren ausgeschlossen. Alle Programmieraufgaben und die Midterm-Klausur werden mit 0 Punkten bewertet.

Es ist erlaubt, Lösungsansätze und Konzepte mit Kommilitonen zu diskutieren. Die eigentliche Erstellung das Codes muss jedoch selbstständig erfolgen. Gruppenarbeit für die Abgaben ist nicht geduldet, und führt ebenso zum Ausschluss aus dem gesamten Bonussystem.

Die Verwendungen von *Google*, *StackOverflow*, etc. ist natürlich erlaubt. Konzepte und/oder Code, welcher von diesen und anderen Quellen übernommen wird, ist dabei eindeutig zu markieren. Bei *StackOverflow* gibt es hierzu unter jeder Antwort einen *share* Link, welcher als Kommentar im Quellcode vermerkt werden muss. Im Zweifelsfall müssen Sie jedoch in der Lage sein, den Code und seine Funktionsweise zu erklären.

Um unnötige Diskussionen zu vermeiden, sei hier auch darauf hingewiesen, dass das Umbenennen von Variablen bzw. Funktionen oder das Umsortieren von Code-Zeilen bzw. Blöcken keine Eigenleistung darstellt, und ebenso wie ein Plagiat behandelt wird.

Anhang: Ein kleines Shell-Einmaleins

`man <Befehlsname>` *manual*
: liefert Hilfe zu einem Befehl; funktioniert auch mit manchen C-Functionen wie `recv`

`pwd` *print working directory*
gibt das aktuelle Verzeichnis aus

`ls` *list directory contents*
listet die Dateien und Ordner im aktuallen Verzeichnis auf

`cd <Verzeichnisname>` *change directory*
wechselt in ein anderes Verzeichnis, z. B.:

- Arbeitsverzeichnis: `/root/01234567`
  Befehl: `cd assignment1`
  wechselt in das Verzechnis `/root/01234567/assignment1`
- Arbeitsverzeichnis: `/root`
  Befehl: `cd /root/01234567/result/assignment1`
  wechselt in das Verzechnis `/root/01234567/result/assignment1`

`make`
kompiliert das Programm, wenn sich im aktuellen Verzeichnis eine Datei namens `Makefile` befindet

`cp <Quelle> <Ziel>` *copy*
eine Datei kopieren; zum Kopieren von Ordner zusätzlich den Parameter „-r“ angeben (`cp -r <Quelle> <Ziel>`)

`mkdir <Ordnername>` *make directory*
einen Ordner erstellen

`rm <Dateiname>` *remove*
eine Datei löschen

`rmdir <Ordnername>` *remove directory*
einen *leeren* Ordner löschen

`mv <Quelle> <Ziel>` *move*
eine Datei oder einen Ordner umbenennen oder verschieben (zum Verschieben einfach bei Ziel einen Pfad mit angeben)

`sudo <Befehl> <Parameter>*` *do as super user*
führt einen Befehl mit den Rechten des Benutzers „root“ aus (andere Betriebssysteme verwenden oft andere Namen wie „super user“ oder „Administrator“); auf der VM haben Sie sich bereits als „root“ angemeldet, also ist der Befehl hier nicht erforderlich

`apt install <Paketname>`
installiert Pakete, also zusätzliche Anwendungen

`./<Programm> <Parameter>*`
führt ein Programm im aktuellen Verzeichnis aus, z. B. `./run -s ../moep8023/moep8023_socket`

`^C` *Tastenkombination: Steuerung+C*
laufendes Programm beenden

`arp -a`
gibt die Einträge im ARP-Cache aus

`ip a`
gibt die vorhandenen Netzwerkadapter und ihre Ethernet- und IP-Adressen aus

`host <Rechnername oder IP-Adresse>`
löst via DNS den angegebenen DNS-Namen in eine IP-Adresse auf (*lookup*) oder löst die angegebene IP-Adresse in einen Namen auf (*reverse lookup*)

`tcpdump -w <Dateiname>`
: solange `tcpdump` läuft, schneidet es den Netzwerkverkehr mit und speichert ihn in der angegebenen Datei

`exit`
: beendet die Sitzung und schließt die Konsole oder trennt die ssh-Verbindung

Ein weiteres nützliches Feature ist Tab Completion: Wenn man gerade einen Pfad eingibt, wie:
`./run ../moe`
kann man die Tab-Taste drücken und erhält als Vorschlag alle möglichen Dateinamen.

Wenn das gerade ausgeführte Programm nicht mehr reagiert, kann man es mit der Tastenkombination `Strg+C` abbrechen.

Aufgabe 1 FizzBuzz — 1 Punkt

Fizzbuzz ist ein einfacher Teaser in Form eines erweiterten „Hello World“. FizzBuzz ist ohne große Kentnisse einer Programmiersprache implementierbar. Die einzigen Sprachkonzepte die man braucht, sind Schleifen und Textausgabe.

a)* Ihre Aufgabe ist es, die Zahlen von 1 bis *X* auf der Standardausgabe zeilenweise auszugeben. Dabei sollen aber alle Zahlen die durch *3* teilbar sind durch `Fizz` und alle Zahlen die durch *5* teilbar sind durch `Buzz` ersetzt werden. Ist eine Zahl sowohl durch *3* als auch durch *5* teilbar, soll diese durch `FizzBuzz` ersetzt werden. Das Limit *X* wird auf der Kommandozeile übergeben.

Das sieht dann zum Beispiel so aus:

```
./fizzbuzz 5
 1
 2
 Fizz
 4
 Buzz
```

Rahmenprogramme

Für C und Java werden im `material`-Repository Rahmenprogramme bereit gestellt. Wenn diese verwendet werden, muss das gesamte Vorlagenordner (`C` bzw. `java`) in das eigene Git-Repositories kopiert und in `assignment1` umbenannt werden. Die interne Struktur muss beibehalten werden. Die zu bearbeiteten Dateien sind im Bezug zum Repository `assignment1/src/assignment1.c` bzw. `assignment1/src/Assignment1.java`.

Automatische Tests

Die im Git abgegeben Programme werden in einer automatischen Testumgebung ausgeführt. Damit diese funktioniert, muss das Verzeichnis `assignment1` in der Wurzel des Repositories verwendet werden.

Die automatischen Tests werden nur dann ausgeführt, wenn Änderungen in dem jeweils aktuellen Assignment-Ordner vorgenommen wurden.

Die Ergebnisse der Testausführung werden nach kurzer Zeit (<5 Minuten) im `result`-Branch im Repository abgelegt. Diese können dann mittels `git pull` vom Server abgerufen werden. Mittels `git checkout result && git pull` kann der Branch lokal ausgecheckt werden. Die Testergebnisse sind nun im Ordner `result/assignment1` zu finden.

Bei erfolgreicher Testausführung enthält `test.log`:

```
Test result:
Testee prints nums: 1
Testee fizzes: 1
Testee buzzes: 1
Testee fizzbuzzes: 1
Testee respecs max: 1
0 minutes and 6 seconds
```

Sollten Tests fehlschlagen werden die Unterschiede vorher aufgezeigt.

Die Dateien `output.log` und `make.log` enthalten wichtigen Debug Output für die Fehlersuche.

Wichtig: Bevor Änderungen am Code vorgenommen werden, muss mittel `git checkout master` wieder auf den `master`-Branch gewechselt werden.

Hinweise falls Sie nicht die Rahmenprogramme verwenden

Es ist möglich, die Aufgaben in einer Sprache Ihrer Wahl zu lösen. Es gibt ein paar Dinge, die dabei aber beachtet werden müssen:

- Der Arbeitsaufwand muss vergleichbar sein mit dem der Rahmenprogramme (magic.fizzbuzz()) währe **nicht** ok. Falls Sie sich unsicher sind, fragen Sie die Übungsleitung.
- Die Ausführbare Datei **muss** *fizzbuzz* heißen.
- Die Ausführbare Datei **muss** die Parameter der Rahmenprogramme unterstützen. (`<max-num>`)
- Es **muss** eine Makefile existieren. Falls Ihre Lösung nicht kompiliert werden muss, da Sie z.B. eine Scriptsprache verwendet haben, muss dennoch eine Dummy-Makefile vorhanden sein, welche ggf. einfach nichts tut.
- Um Pakete vor dem Test zu installieren müssen die Paketnamen in einer Datei *deps.txt* in dem Ordner *assignment1* stehen.
- Pakete dürfen nur aus Debian Jessie *main* und *contrib* installiert werden.

Literatur

[1] *Git – FAQ*. `https://git.wiki.kernel.org/index.php/Git_FAQ`.

[2] *XCode CLI-Tools – Download*. `https://developer.apple.com/library/ios/technotes/tn2339/_index.html`.

[3] *Git for windows – Download*. `https://git-scm.com/download/win`.

[4] *MobarXterm – Download*. `http://mobaxterm.mobatek.net`.

[5] T. Kessler, *How to mount a remote system as a drive using SSH in OS X*. `http://www.macissues.com/2014/10/13/how-to-mount-a-remote-system-as-a-drive-using-ssh-in-os-x/`.

[6] *Git – Commit*. `https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository`.

[7] *Git – Remotes*. `https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes`.