Eine Umgebung zur Unterstützung von Kollaborationen und zur Archivierung von Forschungsergebnissen und -daten am LS4

Falko Bause, Iryna Dohndorf, Jan Kriege, Dimitri Scheftelowitsch
LS Informatik IV, TU Dortmund, 44221 Dortmund
{falko.bause,iryna.dohndorf,jan.kriege,dimitri.scheftelowitsch}@tu-dortmund.de

Zusammenfassung—Das vorliegende Dokument beschreibt die am LS4 eingesetzte Forschungsumgebung. Die Beschreibung umfasst sowohl die Benutzer- als auch die Adminstratorsicht.
Die Forschungsumgebung stützt sich auf git-Repositories [1] ab, deren Zugriff über gitolite [3] kontrolliert wird.

Inhalt

1 Einleitung

In den letzten Jahren gibt es zunehmend Bestrebungen Forschungsergebnisse und zugehörige Daten langfristig verfügbar zu halten, um beispielsweise eine spätere Verwendung und Nachvollziehbarkeit zu gewährleisten [5], [6]. Die hier beschriebene Umgebung soll diese Bestrebungen am LS4 fördern und eine entsprechende Infrastruktur bereitstellen. Desweiteren soll die Zusammenarbeit von Arbeitsgruppen bei der Erarbeitung von Forschungsergebnissen unterstützt werden.

Die Basis der Umgebung bildet eine Menge von git-Repositories[1] in denen Publikationen, Software, Modelle, Rohdaten etc. abgelegt werden. git unterstützt eine verteilte Versionsverwaltung, so dass ggf. auch die (zeitliche) Entwicklung von Forschungsresultaten nachvollziehbar wird.

Um Verfälschungen oder nachträgliche Änderungen der Inhalte zu unterbinden, bedarf es einer geeigneten Zugriffskontrolle. Hierzu wird am LS4 gitolite eingesetzt.

1. Im folgenden wird ein solches git-Repository auch als Kollaborationsumgebung bezeichnet.

Das vorliegende Dokument beschreibt grundlegende Nutzungsmöglichkeiten der Forschungsumgebung und spezifiziert Richtlinien zur Benutzung und Archivierung.

In Kap. 4 wird die Administration der Forschungsumgebung beschrieben. Dieser Abschnitt wurde zum besseren Verständnis in das Dokument aufgenommen, kann aber vom normalen Endanwender (bis auf Unterkapitel 4.8) übersprungen werden.

2 Benutzersicht

2.1 Vorbereitende Tätigkeiten für neue Nutzer

Der Zugriff auf Kollaborationsumgebungen erfolgt über ssh. Nutzer müssen daher ihren **öffentlichen ssh-Schlüssel** via E-Mail an gitadm@ls4 den Adminstratoren der Forschungsumgebung mitteilen.

Im folgenden wird in den entsprechenden `ssh`-Kommandos der Rechnername `rechner` verwendet und bezeichnet einen beliebigen LS4-Linux-Rechner, wie z.B. `ls4hal1` oder `ls4at65`.

2.2 Übersicht über Kollaborationsumgebungen

Zugelassene Nutzer erhalten eine Übersicht via

```
ssh gitadm@rechner info
```

Die Übersicht enthält alle Kollaborationsumgebungen auf die der Nutzer Zugriff hat, inkl. der entsprechenden Rechte (R=Read, W=Write).

2.3 Einrichtung einer Kollaborationsumgebung

Die Einrichtung einer Umgebung erfolgt über eine E-Mail an gitadm@ls4.

In der E-Mail sollten der Name des Repositories angegeben werden, z.B. `lehre/pn_uebung`, sowie die Namen der Personen (und falls abweichend die Namen der Linux-Benutzerkonten), die lesenden bzw. lesenden und schreibenden Zugriff auf das Repository erhalten sollen.

Anmerkung: Um Zugriff zu erhalten, muss der ssh-Schlüssel der entsprechenden Teilnehmer mitgeteilt worden sein, siehe Unterkapitel 2.1.

2.4 Nutzung von Kollaborationsumgebungen

Die Nutzung einer Kollaborationsumgebung erfolgt durch sog. Klonen des git-Repositories aus der Forschungsumgebung und Bearbeiten des lokalen (geklonten) git-Repositories, sowie nachfolgendem Pushen des lokalen git-Repository in die Forschungsumgebung.

Aus abstrakter Sicht ist ist ein git-Repository eine Folge von *Bäumen*, d. h, Dateiinhalten mit assoziierten Pfaden. Ein neuer Baum entsteht durch *Commits*, ausgezeichnete ("'abgesegnete"') Änderungen der Inhalte (etwa, Änderung oder Hinzufügen einer Datei). Bäume (und damit auch Commits) werden mit Prüfsummen (in Form einer langen Hexadezimalzahl) identifiziert.

Da git nicht zentral verwaltet wird, kann es sein, dass mehrere Benutzer verschiedene Folgen von Commits als "'Geschichte"' des Repositorys sehen, selbst wenn der aktuelle Zustand der gleiche ist. An dieser Stelle sei herausgestellt, dass dies kein Mangel, sondern notwendige Folge einer dezentralen Versionsverwaltung ist.

Ein git-Repository besteht i.w. aus drei Bereichen: einem Arbeits-Ordner mit allen Dateien, dem sog. staging-Bereich und einem commit-Bereich. Im Arbeitsordner werden die Dateien bearbeitet. Mittels `git add DATEI_1 .. DATEI_N` werden die Dateien in den staging-Bereich transferiert. Die Dateien im staging-Bereich (in exakt der Version des letzten `git add`-Befehls) werden bei einem `git commit`-Befehl in den commit-Bereich transferiert. Ein nachfolgender `push`-Befehl in die LS4-Forschungsumgebung transferiert genau diesen commit-Bereich (in exakt der Version des letzten `git commit`-Befehls).

Der `git commit`-Befehl erfordert die Angabe eines (möglichst aussagekräftigen) Kommentars mittels z.B. der Option `-m`. Falls keine Option angegeben wird, wird ein Editor geöffnet und die Ausführung des commit-Befehls erfolgt nach Speichern des Editor-Inhaltes.

Zu weiteren Informationen hinsichtlich der Nutzung von git siehe [1], [2].

2.4.1 *Wichtige Befehle*

Abbildung 1 enthält eine Übersicht der wichtigsten Befehle.

Hinweis: Beim Klonen wird der Ordner `<REPO-TITEL>` angelegt.[2] Der Klone-Befehl erfordert die Angabe des kompletten Pfades, z.B `lehre/mao_uebung`, der Name des Ordners ist in diesem Fall `mao_uebung`. Die weiteren git-Befehle werden innerhalb dieses Ordners ausgeführt.

3 Richtlinien zur Benutzung

Damit eine Kollaborationsumgebung für andere Benutzer nutzbar ist, sollten bestimmte Grundzüge

2. Der Befehl *git pull* innerhalb eines bereits existierenden Ordners fügt Änderungen aus der Forschungsumgebung in das lokale Repository ein.

- Übersicht aller Repositories auf die ein Nutzer zugreifen kann:
  `ssh gitadm@rechner.cs.tu-dortmund.de info`
- Auschecken eines Repositories:
  `git clone gitadm@rechner.cs.tu-dortmund.de:<REPO-PFAD>`
- Hinzufügen von neuen Dateien:
  `git add <DATEINAME>`
- Commit von Änderungen in das lokale Repository:
  `git commit -m "<KOMMENTAR>"`
- Änderungen aus dem lokalen Repository hochladen:
  `git push origin master`

Abb. 1: Wichtige Befehle zur Nutzung der LS4-Forschungsumgebung

hinsichtlich Ordnerstruktur und Dateiinhalten beachtet werden. So ist es z.B. ratsam im Hauptordner eine README-Datei anzulegen[3], sowie bei Software, LaTeX-Dateien etc. entsprechende Makefiles, die eine Erstellung der ausführbaren Programme oder PDF-Dokumente steuern.

Im folgenden werden einige Hinweise hinsichtlich einzelner Kollaborationsumgebungs-Typen gegeben.

3.1 Veröffentlichungen

Repositories für Veröffentlichungen folgen der Namenskonvention `pub/<REPO-TITEL>`. Der Hauptordner sollte eine README-Datei enthalten mit der Angabe der Veröffentlichung, vgl. Abb. 2.

```
LaTeX-Sourcen zur Veroeffentlichung:

Falko Bause, Jan Kriege:
Correlated Random Number Generation
for Simulation Experiments.
Proc. of the ASIM Conference
on Simulation in Production and
Logistics, 2015.
```

Abb. 2: Beispiel einer README-Datei

Ferner sollte z.B. bei LaTeX-Dateien ein Makefile vorhanden sein (vgl. Abb. 3) über welches die finalen PDF-Dateien o.ä. generiert werden können.

3. Für README-Dateien im ASCII-Format existiert ein sog. Markdown-Format, welches den Vorteil bietet relativ einfach in ein anderes Format umgewandelt werden zu können, wie z.B. in PDF, HTML, oder LaTeX. Eine Beschreibung der Syntax findet der Leser unter [7] und ein Tool zur Konvertierung ist beispielsweise Pandoc [8].

Unterordner sollten nach Möglichkeit die verwendeten Programme (Software), Modelle, Ergebnisse etc. enthalten. Weitere README-Dateien in diesen Unterordner sind auch hier sicherlich für zukünftige Nutzer hilfreich, z.B. mit Informationen zu Aufrufparametern der verwendeten Tools oder Herkunft von Eingabedaten für Modelle.

```
sla_paper:
        pdflatex sla_paper.tex
        bibtex sla_paper
        pdflatex sla_paper.tex
        pdflatex sla_paper.tex
clean:
        rm -f *.aux; rm -f *.dvi
```

Abb. 3: Beispiel eines Makefiles für LaTeX-Artikel

```
pub/<REPO-TITEL>
├── Paper
│   └── Verwendete_Ergebnisse
└── Experimente
    ├── Software
    ├── Modelle
    └── Ergebnisse
```

Abb. 4: Empfohlene Ordnerstruktur für Veröffentlichungen

3.2 Software

Mindestens eine README-Datei im Hauptordner mit einer kurzen Beschreibung der Software,

Angabe von zu beachtenden Lizenzen, Lizenz dieser Software etc.

```
software/<REPO-TITEL>
├── Sourcen
├── Dokumentation
│   ├── Manual
│   └── Installation
├── Executables
└── Beispiele
```

Abb. 5: Empfohlene Ordnerstruktur für Software

3.3 Modelle

README-Datei im Hauptordner.

```
models/<REPO-TITEL>
├── Modelle
└── Dokumentation
```

Abb. 6: Empfohlene Ordnerstruktur für Modelle

3.4 Rohdaten

README-Datei im Hauptordner.

```
data/<REPO-TITEL>
├── Daten
└── Dokumentation
```

Abb. 7: Empfohlene Ordnerstruktur für Rohdaten

3.5 Lehrveranstaltungen

Repositories für Lehrveranstaltungen folgen der Namenskonvention `lehre/<Name der Veranstaltung>` (z.B. `lehre/rvs_uebung`). Das Repository enthält für jedes Jahr, in dem die Veranstaltung stattfand, ein Unterverzeichnis mit Übungsblättern und sonstigem Material:

Unterverzeichnisse für neue Jahre können von den Nutzern eigenständig angelegt werden.

```
lehre/<REPO-TITEL>
├── <JAHR 1>
│   ├── Uebungsblaetter
│   └── Sonstiges
└── <JAHR 2>
    ├── Uebungsblaetter
    └── Sonstiges
```

Abb. 8: Empfohlene Ordnerstruktur für Lehrveranstaltungen

4 Administration

Die Nutzung der LS4-Forschungsumgebung erfolgt über die Software gitolite [3], [4]. Ein spezieller Nutzer am LS4 (`gitadm`) hat exklusiven Zugriff auf alle Dateien, die zur Forschungsumgebung gehören. Hierzu gehören neben Konfigurationsdateien und allen Repositories, auch die öffentlichen ssh-Schüssel aller Nutzer der Forschungsumgebung.[4]

Die Administration der Forschungsumgebung erfolgt über ein spezielles Repository `gitolite-admin` auf welches der Zugriff genau wie auf andere Repositories erfolgt, sofern der Nutzer die entsprechenden Rechte besitzt (siehe `ssh gitadm@rechner info`). Durch `git clone gitadm@rechner:gitolite-admin` wird das Repository zur Administration geklont und es wird der lokale Ordner `gitolite-admin` angelegt. Dieser Ordner enthält zwei Unterordner `conf` und `keydir`, vgl. Abb. 9.

Der Ordner `keydir` enthält alle öffentlichen ssh-Schlüssel der Nutzer der LS4-Forschungsumgebung. Die Dateinamen müssen hierbei dem Muster `<Linux-Accountname>.pub` folgen.

Die Datei `gitolite.conf` ist die zentrale Konfigurationsdatei der Forschungsumgebung. Hier werden Repositories, Zugriffsrechte, Nutzer und Nutzergruppen definiert. Kommentarzeilen werden durch das Zeichen # eingeleitet. Die Einträge werden durch gitolite in der Reihenfolge ihres Auftretens berücksichtigt [3].

Dies sollte bei der Definition unbedingt beachtet werden. Beispiel aus [4]:

```
@developers     =   dilbert alice
```

4. Nach Erstinstallation der Forschungsumgebung muss der explizite Zugriff auf das Nutzerkonto *gitadm* nur in Ausnahmefällen erfolgen.

```
.:
total 6
drwx------ 2 bause ls4 3 Feb 21 09:37 conf
drwx------ 2 bause ls4 6 Feb 21 09:37 keydir

./conf:
total 2
-rw------- 1 bause ls4 1049 Feb 21 09:37 gitolite.conf

./keydir:
total 9
-rw------- 1 bause ls4 223 Feb 21 09:37 bause.pub
-rw------- 1 bause ls4 395 Feb 21 09:37 felko.pub
-rw------- 1 bause ls4 736 Feb 21 09:37 kriege.pub
-rw------- 1 bause ls4 628 Feb 21 09:37 scheftel.pub
...
```

Abb. 9: Beispiel von Ordnerinhalten des Administrationsrepositories

```
@interns        =   ashok
@staff          =   @interns @developers
@developers     =   wally

# wally is NOT part of @staff
# but part of developers which
#       are dilbert alice wally
```

Alle Änderungen in dem lokalen `gitolite-admin` Repository werden erst aktiv, wenn dieses Repository "gepusht" worden ist. Die folgende beispielhafte Abfolge von Befehlen beschreibt den typischen Ablauf von Administrationstätigkeiten:

```
git clone gitadm@rechner:gitolite-admin
cd gitolite-admin
<ggf. Hinzufuegen von Schluesseln im
                    Ordner keydir>
<Editieren der Datei gitolite.conf>
git add *  (aus dem lokalen Ordner
                  gitolite-admin!)
git commit -m "<Kommentar>"
git push origin master
```

4.1 Einrichtung einer Kollaborationsumgebung

Die Einrichtung erfolgt durch Eintragung des Namens der Kollaborationsumgebung, sowie der Nutzer bzw. Nutzergruppen und deren Rechte in der Datei `gitolite.conf`. `gitolite` legt mit dem Push-Befehl intern ein sog. "bare git repository" an.

Beispiel:

```
...
# Repo zur Octave Programmierung
repo pgm/octavesla
    RW+     =   @rebauf felko
    R       =   scheftel
...
```

4.2 Löschen einer Kollaborationsumgebung

Ist nicht vorgesehen, siehe Abschnitt 4.8.

4.3 Gruppen von Kollaborationsumgebungen

Eine Gruppe von Kollaborationsumgebungen kann durch Angabe der einzelnen Repositories (in der Datei `gitolite.conf`) definiert werden. Der Gruppenname kann durch ein vorangestelltes `@` innerhalb der Konfigurationsdatei genutzt werden.

Beispiel:

```
...
@staff     =   dilbert george # groups
@projects  =   foo bar

repo @projects baz          # repos
    RW+ =   @staff michel   # rules
...
```

Hinweis: gitolite besitzt eine standardmäßig definierte Gruppe `all`, welche alle Repositories umfasst.

4.4 Hinzufügen von Benutzern

Das Hinzufügen von Benutzern erfordert zum einen das Abspeichern des öffentlichen ssh-Schlüssels im Ordner `keydir` und zum anderen die Eintragung des Benutzer in die Datei `gitolite.conf`.

Die Schlüsseldatei muss zwingend der Namenskonvention `<Linux-Accountname>.pub` folgen.

Beispiel für eine Eintragung in die Datei `gitolite.conf`:

```
...
@staff      =   dilbert george # groups
@extern     =   michel new_user
@projects   =   foo bar

repo @projects baz          # repos
    RW+ =   @staff michel   # rules
     R  =   new_user
```

4.5 Löschen von Benutzern

Eine Löschung erfolgt durch Löschen der zugehörigen Schlüssel-Datei im Ordner `keydir` und durch Austragen des Benutzernamens aus der Konfigurationsdatei `gitolite.conf`.

4.6 Benutzergruppen

Eine Gruppe von Benutzern wird durch Auflistung der zugehörigen Nutzer in der Datei `gitolite.conf` definiert. Nachfolgend kann der Gruppenname bei der Definition von Zugriffsrechten durch Angabe eines vorangestellten `@` verwendet werden.

Hinweis: gitolite besitzt eine standardmäßig definierte Gruppe `all`, welche alle Benutzer umfasst.

4.7 Zugriffsrechte

gitolite kennt folgende Zugriffsrechte:

- R — erlaubt lesenden Zugriff (“clone” und “fetch” sind erlaubt, aber keine “push”-Operationen).
- RW — erlaubt zusätzlich auch schreibenden Zugriff (sog. “fast-forward push”-Operationen und das Anlegen neuer “branches” ist erlaubt).
- RW+ — erlaubt zusätzlich sog. “non-fast forward pushes” und auch Löschungen von “branches”.
- \- — verweigert explizit den Zugriff.
- C — erlaubt das Anlegen von Repositories. Beispiel:

```
repo pub/CREATOR/..*
    C       =   bill heinz
    RW+     =   CREATOR
    RW      =   user1 user2
    R       =   user3
```

  Erlaubt den Nutzern `bill` und `heinz` das Anlegen eigener Kollaborationsumgebungen. Der Name der Umgebung ist in diesem Fall `pub/<Linux-Accountname>`.
- D — erlaubt das explizite Löschen.

Hinweis: Die Rechte `C,D,-` sollten bei der Konfiguration der LS4-Forschungsumgebung nur in Ausnahmefällen verwendet werden.

4.8 Archivierung

Die Archivierung einer Kollaborationsumgebung erfolgt auf Anforderung der Nutzer und muss dem Administrator mitgeteilt werden. Hierbei werden für das entsprechende Repository alle schreibenden und erweiterten Rechte (`W+`) in der Datei `gitolite.conf` entfernt.

Die Nutzer einer Kollaborationsumgebung sollten vor Archivierung sicherstellen, dass das Repository entsprechende README-Dateien und Dateien zur Dokumentation enthält um die Nachvollziehbarkeit von Forschungsresultaten zur gewährleisten.

Anmerkung: Um das Löschen von alten Inhalten zu verhindern, sollte im Regelfall nur das Recht `RW` anstelle von `RW+` bei der Einrichtung einer Kollaborationsumgebung verwendet werden.

Literatur

[1] S. Chacon, B. Straub: Pro Git. Apress, 2014.

[2] git - Der einfache Einstieg. https://rogerdudler.github.io/git-guide/index.de.html

[3] S. Chamarty: Gitolite Essentials. Packt Publishing, 2014.

[4] gitolite all-in-one page. http://gitolite.com/gitolite/gitolite.html

[5] Umgang mit digitalen Daten in der Wissenschaft: Forschungsdatenmanagement in NRW – Eine erste Bestandsaufnahme, DV-ISA, 2016. https://www.dh-nrw.de/fileadmin/dh-nrw/PDF/Veroeffentlichungen/DV-ISA-Bestandsaufnahme_FDM.pdf

[6] Empfehlungen zur gesicherten Aufbewahrung und Bereitstellung digitaler Forschungsprimärdaten, DFG, 2009. http://www.dfg.de/download/pdf/foerderung/programme/lis/ua_inf_empfehlungen_200901.pdf

[7] Markdown: Basics. https://daringfireball.net/projects/markdown/basics

[8] Pandoc – a universal document converter. http://pandoc.org

```
# Nutzergruppen
# -------------

@lehrende  =  felko scheftel
@rebauf    =  bause kriege
@wimis     =  bause felko kriege scheftel

# Repos
# -----

# gitolite Administration
repo gitolite-admin
    RW+     =   @rebauf

# Dokumentation der LS4-Repos
repo ls4forschungsumgebung
    RW+     =   @rebauf felko scheftel
    R       =   @all


# Lehre
# -----

# Dokumente zur Durchfuehrung von Klausuren
repo lehre/klausurvorbereitung
    RW+     =   @rebauf
    R       =   @lehrende

# Uebung MAevS (Modellierung und Analyse eingebetteter und vert. Systeme)
repo lehre/maevs_uebung
    RW+     =   @rebauf felko
    R       =   @lehrende

# Uebung MAO (Modellgestützte Analyse und Optimierung)
repo lehre/mao_uebung
    RW+     =   @rebauf felko
    R       =   @lehrende

# Uebung Mathe 2
repo lehre/mathe2_uebung
    RW+     =   @rebauf scheftel
    R       =   @lehrende

# Uebung RvS (Rechnernetze und verteilte Systeme)
repo lehre/rvs_uebung
    RW+     =   @rebauf scheftel
    R       =   @lehrende
...
```

Abb. 10: Beispiel einer *gitolite.conf* Datei