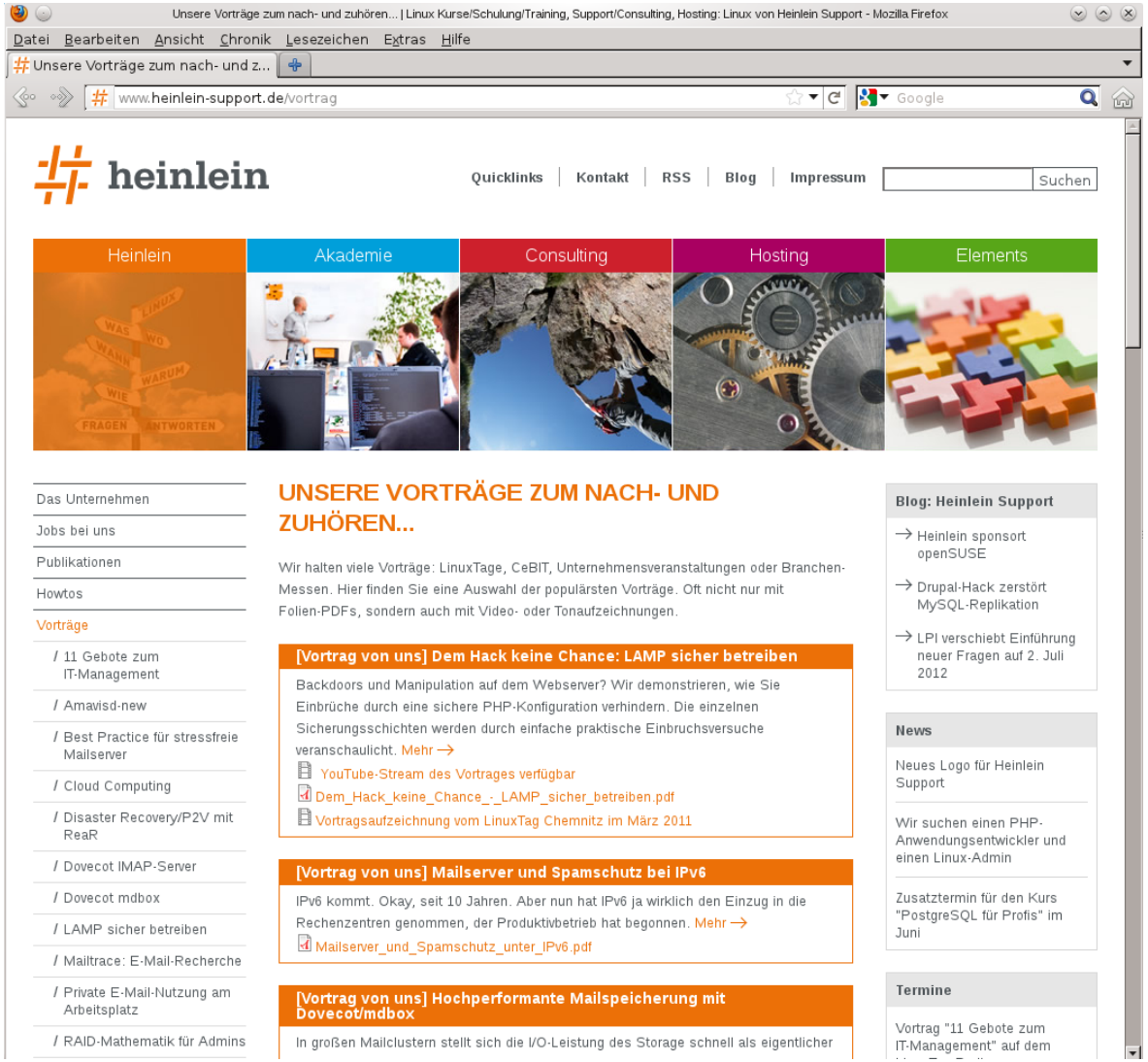
Silke Meyer <s.meyer@heinlein-support.de>

Git für Admins

heinlein

Silke Meyer <s.meyer@heinlein-support.de>



Ja, diese Folien stehen auch als PDF im Netz...
https://www.heinlein-support.de/vortrag

Silke Meyer <s.meyer@heinlein-support.de>

Einsatzbereiche von git in der Systemadministration

- Kollaboratives Arbeiten an allen Arten von (textbasierten) Inhalten
  - Texte, Dokumentation
  - Code, Skripte
  - Konfigurationsdateien
- Für Installation und Updates von Software (statt .tar.gz)
- Als Backup für die eigene Arbeit

Git-Plattformen

- Git-Server selbst hosten
  - Gitlab (freie Software in Ruby, MIT-Lizenz), mit Jenkins/Gitlab CI z.B. für Debian-Paketbau
  - Phabricator (integrierter Repo-Browser, Bugtracker, Code Review, Wiki, in PHP, Apache-Lizenz)
  - klein: Gogs (in Go, Lizenz unklar, aber wenig restriktiv)
  - proprietäre / kostenpflichtige Software „Bitbucket Server“ von Atlassian
- SaaS-Möglichkeiten sind u.a.
  - gitlab.com
  - github.com (nur private Repos kostenpflichtig)
  - bitbucket.com (Atlassian, bis 5 Accounts kostenlos, auch private Repos)

Der Arbeitsablauf

- Konzept machen, Absprachen treffen im Team
- Repository(s) aufsetzen mit dem aktuellen Stand der Dateien
- Eine eigene Kopie des Repositorys runterladen
- Darin Änderungen vornehmen
- Änderungen lokal in git speichern
- Änderungen hochladen
- Ggf. Pull Request stellen in den abgesprochenen Branch

Lokale Konfiguration

- `aptitude install git(-core)`
- Konfigs systemweit / pro Useraccount / pro Git-Repo
- `ich@tux:~/testrepo $ cat ~/.gitconfig`

```
[user]
    name = Silke Meyer
    email = s.meyer@heinlein-hosting.de
[core]
    editor = vim
[merge]
    tool = meld
```

Lokale Konfiguration

- Upload (git push) per ssh vorbereiten:
  - Webinterface: ssh public key hochladen
  - `ich@tux:~/testrepo $ cat ~/.ssh/config`
    ```
    Host = github.com
    IdentityFile /home/ich/.ssh/mykey
    ```
  - `ich@tux:~/testrepo $ ssh -T git@github.com`
    ```
    Hi silke! You've successfully authenticated, but GitHub does not
    provide shell access.
    ```
- Protipp: aktuellen Branch im Shell Prompt anzeigen lassen
  - `ich@tux:~/testrepo (master) $`
  - Link zu Shell-Integration für git

Commit

- Eine in git gespeicherte Änderung
  - Der Zustand des ganzen Repos zu einem Zeitpunkt
  - Metadaten im git log: eindeutige Commit-ID, Autor, Zeitstempel, Commit Message
- `ich@tux:~/testrepo $ git log`

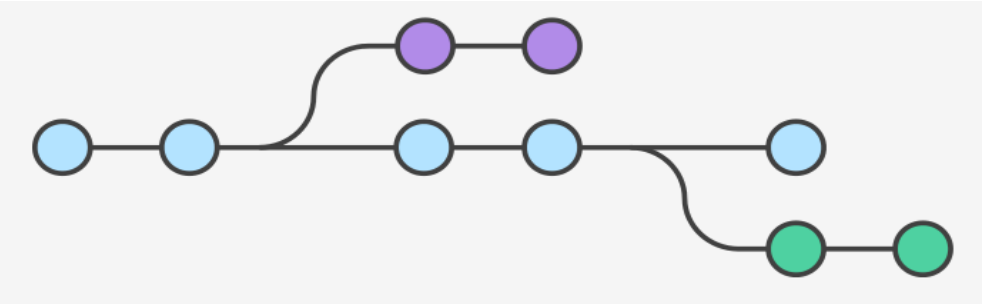
```
commit 55bc83dffc398ffe8e1466262993c8c1bd342f64
Author: Silke Meyer <silke@silkemeyer.net>
Date:   Tue Mar 15 12:11:13 2016 +0100

    Abkürzung ausgeschrieben
```

Silke Meyer <s.meyer@heinlein-support.de>

Branches

- „master“ (der Hauptzweig)
- Eigene Branches mit beliebigen Namen („development“, „feature“)
- `ich@tux:~/testrepo $ git checkout -b veranstaltungsinfos`
  `Switched to a new branch 'veranstaltungsinfos'`
- Absprache im Team: Wie arbeiten wir mit Branches? (git-Workflows)

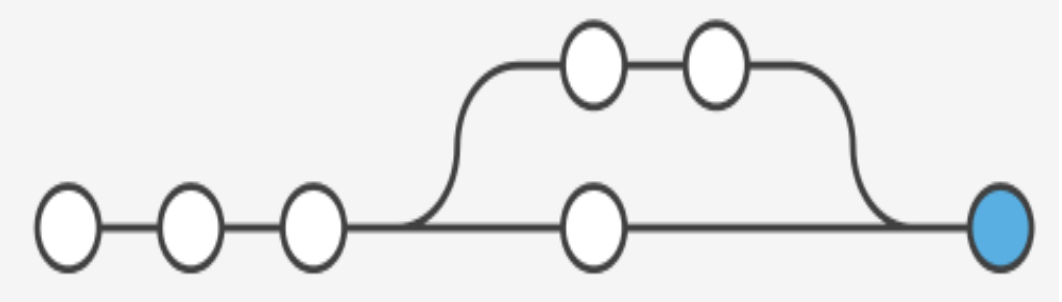


https://www.atlassian.com/git/tutorials/using-branches

Silke Meyer <s.meyer@heinlein-support.de>

Mergen

- Das Zusammenführung von Branches
- Einen anderen Branch in den aktuellen Branchen „mergen“
- ```
  ich@tux:~/testrepo $ git checkout master
  ich@tux:~/testrepo $ git merge veranstaltungsinfos
  ```
- Pull Request: „Bitte meine Änderung aufnehmen/mergen!“



https://www.atlassian.com/git/tutorials/using-branches/git-merge
  ```

Silke Meyer <s.meyer@heinlein-support.de>

Orte und Zustände

- Serverseitig
  - Origin: meine Kopie des Projekt-Repos
  - Upstream: optional, ein Repo, von dem Origin abhängt
- Lokal
  - Lokales Repository: der lokale Klon des Origin plus meine lokalen (nicht hochgeladenen) Commits
  - Staging Area / Index: alles, was ich lokal in meinen nächsten Commit aufnehmen möchte / was ich unter Versionskontrolle stelle (git add <datei>)
  - Working Copy: meine Arbeitsdateien, inkl. aller uncommitteter Änderungen und Branches, auch Dateien im Verzeichnis, die nicht unter Versionskontrolle stehen

Silke Meyer <s.meyer@heinlein-support.de>

Nochmal: Die Schritte eines Commits

- Kopie des Originalrepo runterladen (git clone <url>)
- einen Branch anlegen nach Absprache (git checkout -b <name>)
- Änderungen vornehmen
- Änderungen für den nächsten Commit vormerken (git add <dateien>)
- Committen (git commit -m „Kommentar“)
- Hochladen (git push origin <branchname>)
- Pull request stellen (im Webinterface)

Das erste Herunterladen

```
→ ich@tux:~/$ git clone
  git@github.com:silke/testrepo.git
  Cloning into 'testrepo'…
  remote: Counting objects: 4, done.
  remote: Compressing objects: 100% (3/3), done.
  remote: Total 4 (delta 0), reused 4 (delta 0), pack-
  reused 0
  Receiving objects: 100% (4/4), 12.14 KiB | 0 bytes/s,
  done.
  Checking connectivity... done.
→ ich@tux:~/$ cd testrepo
→ ich@tux:~/testrepo (master) $
```

Branch anlegen

```
→ ich@tux:~/testrepo $ git branch -v
  * master                  b3e394c Initial commit
    ...                       ...
→ ich@tux:~/testrepo $ git checkout -b
  veranstaltungsinfos
  Switched to a new branch 'veranstaltungsinfos'
→ ich@tux:~/testrepo $ git branch -v
    master                  b3e394c ...
  * veranstaltungsinfos     b3e394c
    ...                       ...
→ ich@tux:~/testrepo $ git checkout master
  Switched to branch 'master'
```

Nach Änderung: Status prüfen

```
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git status
  On branch veranstaltungsinfos
  Changes not staged for commit:
    (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in
  working directory)

  modified:   README.md

  Untracked files:

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

  programm.txt
```

Diff prüfen

```
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git diff
  diff --git a/README.md b/README.md
  index edb7f85..6201877 100644
  --- a/README.md
  +++ b/README.md
  @@ -1,2 +1,5 @@
   # testrepo
   Testrepo für die SLAC 2016
  +
  +Datum: 15.-17. Juni 2016
  +Ort: Berlin, irgendwo in der West-City
```

Silke Meyer <s.meyer@heinlein-support.de>

Dateien ins Staging aufnehmen

```
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git add README.md
  programm.txt
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git status
  On branch veranstaltungsinfos
  Changes to be committed:
    (use "git reset HEAD <file>..." to unstage)

  modified:   README.md
  new file:   programm.txt
```

Ins lokale Repo committen

```
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git commit -m „mehr infos zur SLAC“
[veranstaltungsinfos 2751fb3] mehr infos zu der SLAC
 2 files changed, 3 insertions(+)
 create mode 100644 programm.txt
→ ich@tux:~/testrepo (veranstaltungsinfos)$ git log
commit 2751fb3c6b7b7a0c1739a56666e38b7de04b0168
Author: Silke Meyer <silke@silkemeyer.net>
Date:   Tue Mar 15 12:43:52 2016 +0100

    mehr infos zur SLAC

...
```

Ins Origin-Repo hochladen

- Der lokale Branch soll ins entfernte Repository.
- **ich@tux:~/testrepo (veranstaltungsinfos)$ git push origin veranstaltungsinfos**

```
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 715 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
To git@github.com:silke/testrepo.git
 * [new branch]      veranstaltungsinfos -> veranstaltungsinfos
```

Der Pull Request

- Extra-Tools für Pull Requests von Kommandozeile
- Einfacher: Ab ins Webinterface und Button betätigen! ;)

Merge-Konflikte

- Beispiel: README.md wurden in beiden lokalen Branches (master und veranstaltungsinfos) bearbeitet.
- ```
  ich@tux:~/testrepo (master)$ git merge veranstaltungsinfos
  Auto-merging README.md
  CONFLICT (content): Merge conflict in README.md
  Automatic merge failed; fix conflicts and then commit the result.
  ```
- Die konfligierende(n) Datei(en) werden markiert.
  ```

Merge-Konflikte

```
→ ich@tux:~/testrepo (master)$ cat README.md
  # testrepo
  <<<<<<< HEAD
  Testrepo für die SLAC 2016
  Wann? 15.-17.6.2016
  Wo? Irgendwo in der West-City
  Hashtag: TBA
  =======
  Veranstaltungsankündigung für die SLAC
  Datum: 15.-17. Juni 2016
  Ort: Hotel in Berlin-Tiergarten
  >>>>>>> veranstaltungsinfos
```

Silke Meyer <s.meyer@heinlein-support.de>

Merge-Konflikt lösen

- Abbrechen…?
  - `ich@tux:~/testrepo (master)$ git merge --abort`
- Manuell nachbearbeiten
  - `ich@tux:~/testrepo (master)$ git mergetool`

```
Merging:
README.md
Normal merge conflict for 'README.md':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

- Editor lädt drei Versionen der Datei
  - Alte Version (vor dem Konflikt)
  - Neue Version im aktuellen Branch („local“)
  - Neue Version im reingeholten Branch („remote“)

Waaah! Kopf ab?

- Einen Commit auschecken: auf *keinem* Branch sein.
-
```
ich@tux:~/testrepo (master)$ git checkout
14d767e35c85b2eb14d3c0eb077309dd3c413848
Note: checking out
'14d767e35c85b2eb14d3c0eb077309dd3c413848'.
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 performing another checkout.
```
- Zurück auf einen Branch gehen mit `git checkout <branchname>`

Lokales Repo aus Upstream aktualisieren

```
→ ich@tux:~/testrepo (master) $ git remote -v
  origin   git@github.com:silke/testrepo.git (fetch)
  origin   git@github.com:silke/testrepo.git (push)

→ ich@tux:~/testrepo (master) $ git remote add upstream
   https://github.com/SilkeMeyer/testrepo.git

→ ich@tux:~/testrepo (master) $ git remote -v
  origin   git@github.com:silke/testrepo.git (fetch)
  origin   git@github.com:silke/testrepo.git (push)
  upstream https://github.com/SilkeMeyer/testrepo.git
  (fetch)
  upstream https://github.com/SilkeMeyer/testrepo.git
  (push)
```

Lokales Repo aus Upstream aktualisieren

```
→ ich@tux:~/testrepo (master)$ git fetch upstream
  remote: Counting objects: 3, done.
  remote: Compressing objects: 100% (3/3), done.
  remote: Total 3 (delta 0), reused 3 (delta 0), pack-
  reused 0
  Unpacking objects: 100% (3/3), done.
  From https://github.com/SilkeMeyer/testrepo
  * [new branch]      master     -> upstream/master

→ ich@tux:~/testrepo (master) $ git merge
  upstream/master
  Updating b3e394c..55bc83d
  Fast-forward
  README.md | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)
```

Tags

- Markierte Commits, z.B. für Release oder Release Candidate
- Tag erstellen:
  ```
  ich@tux:~/testrepo $ git tag -a v1.0 -m "Release v1.0" 8d5df3bd0998508403c1eaac240e26fa5462b24a
  ```
- Auf getaggten Commit wechseln:
  ```
  ich@tux:~/testrepo $ git checkout v1.0
  ```
- Lokal erstellte Tags hochladen:
  ```
  ich@tux:~/testrepo $ git push --tags
  ```

Git-Workflows

- Absprachen zum Release-Management
- Absprachen zum Umgang mit Branches
- Dokumentierte Erfahrungsberichte anderer lesen und ggf. adaptieren, z.B. „A successful git branching model“ (Vincent Driessen)

Review-Tools

- Vier Augen sehen mehr als zwei!
- Absprache: Wer darf mergen? Dürfen alle selbst mergen?
- Webinterfaces bieten i.d.R. Auswahl von Reviewern an.
- Extra-Tools: gerrit → erweiterter Konfigurationsmöglichkeiten

Einsatzszenario I: Nie wieder aus tarballs updaten

- der Klassiker: unpraktische Softwareupdates mit Auspacken von tar-Archiven, hin- und herkopieren von Ordnern
- Stattdessen: Datenbank sichern und git pull
- Eignet sich vor allem, wenn keine/wenig eigene Änderungen an der Software vorgenommen wurden, in eigenem Branch
- Beispiel: MediaWiki mit Extensions als git submodules holen und in Einklang bringen

Silke Meyer <s.meyer@heinlein-support.de>

Einsatzszenario II: Teamwork an Konfigurationsdateien

- Git-Server mit klarem Workflow im Team
- Der Softwareentwicklung die praktischen Werkzeuge abgucken
- Entwicklungs-/Testumgebung und Produktivsysteme…
- … mit verschiedenen Branches z.B. „testing“, „production“ etc.
- Nutzen:
  - Weiterentwicklung von Skripten und Konfigs verfolgen
  - Fehler zurückrollen können / Risikominimierung
  - Nicht beabsichtigte Änderungen möglichst mitbekommen (git diff)
  - Leichteres Testen
  - Änderungen Personen zuordnen können → wen fragen, wenn…?

Lektüre

- Offizielle Doku
- Valentin Haenel / Julius Plenz: Das Git-Buch, Open Source Press, 2014 (CC BY-NC-SA 4.0)
- Scott Chacon / Ben Straub: Pro Git, Apress, 2009 (CC BY-NC-SA 3.0)
- Seth Robertson: On undoing, fixing, or removing commits in git
- Vincent Driessen: A successful Git branching model
- Christie Koehler: Fun with git submodules
- Offizielle Github Hilfe, Githubs Cheatsheet
- Sucht Euch Online-Tutorials!

- Danke für das Interesse!
- Natürlich und gerne stehe ich Ihnen jederzeit mit Rat und Tat zur Verfügung und freue mich auf neue Kontakte.
  - Silke Meyer
  - Mail: s.meyer@heinlein-support.de
  - Telefon: 030/40 50 51 - 51
- Wenn's brennt:
  - Heinlein Support 24/7 Notfall-Hotline: 030/40 505 - 110

Silke Meyer <s.meyer@heinlein-support.de>

Wir suchen:

Admins, Consultants, Trainer!

Wir bieten:

Spannende Projekte, Kundenlob, eigenständige Arbeit, keine Überstunden, Teamarbeit

...und natürlich: Linux, Linux, Linux...

http://www.heinlein-support.de/jobs

Silke Meyer <s.meyer@heinlein-support.de>

heinlein

Heinlein Support hilft bei allen Fragen rund um Linux-Server

HEINLEIN AKADEMIE

Von Profis für Profis: Wir vermitteln die oberen 10% Wissen: geballtes Wissen und umfangreiche Praxiserfahrung.

HEINLEIN HOSTING

Individuelles Business-Hosting mit perfekter Maintenance durch unsere Profis. Sicherheit und Verfügbarkeit stehen an erster Stelle.

HEINLEIN CONSULTING

Das Backup für Ihre Linux-Administration: LPIC-2-Profis lösen im CompetenceCall Notfälle, auch in SLAs mit 24/7-Verfügbarkeit.

HEINLEIN ELEMENTS

Hard- und Software-Appliances und speziell für den Serverbetrieb konzipierte Software rund ums Thema eMail.