git – versioncontrol . . .

Johannes Hubertz

hubertz-it-consulting GmbH

TroLUG – Troisdorf, 3. April 2014

Vorstellung: Johannes Hubertz

1980 Hardware-Reparatur: Ersatzteile auf Bauteilebene
1984 Entwicklung Sonderprodukte, Assembler, PLM
1987 Erstkontakt mit Unix (SCO-Xenix) und C
1994 Erstkontakt mit IP
1996 Xlink, root@www.bundestag.de, . . .
1997 SSLeay, ipfwadm mit shell-scripts
1998 „Ins Allerheiligste“, iX 1/1998, Heise Verlag
1999 IT-Security Mgr. D-A-CH
2001 Gibraltar, FreeSwan, iptables . . .
2001 Firmenteilung $\rightarrow$ Entwicklung und Betrieb sspe
2002 Weiterentwicklung und Betrieb von sspe
2005 Gründung der hubertz-it-consulting GmbH
2013 **Netzwerk mit Schutzmaßnahmen**, Lehmanns Media GmbH
seit 2001 Segeln, am liebsten auf Salzwasser
seit 2008 Mitarbeit im Vorstand des GUUG e.V.

Vorstellung: hubertz-it-consulting GmbH

Erkenntnisse aus dem Berufsleben

Bellovin and Cheswick: Firewalls and Internet Security, 1994

Fazit: Keep it simple!

Oder mit Einstein: So einfach wie möglich, aber nicht einfacher!

Etwas Erfahrung war Voraussetzung

Gründung am 8. August 2005, Sitz in Köln

Geschäftsinhalt: Dienstleistungen im Umfeld der IT-Sicherheit

Logo: Johannes Hubertz Certificate Authority als ASCII-7Bitmuster

Diese paar Bits findet sich in einigen 10000 X.509 Anwenderzertifikaten in der Seriennummer wieder

Wir sind käuflich ;-)

Portfolio

Netzwerk IPv4 und IPv6

Linux für Firewalls und VPN (IPsec, StrongSwan, OpenVPN, x.509)

Linux als Rendevous-Server mit OpenSSH

Switches: bridge-utils, OpenVSwitch, Cisco™, . . .
Switches: Spanning Tree, VLAN, Channels, Bonding, LACP, . . .

Router: Linux, Quagga, Cisco™, . . .
Router: statisch, dynamisch, OSPF, OSPFv3, radvd, . . .

Virtualisierung

Server: Xen, KVM, Clusterbau, Heartbeat, drbd, ISCSI . . .

Firewalls: iptables ⇒ sspe, ip6tables ⇒ adm6

Netzwerk: pox, OpenVSwitch, Hardwareswitches, OpenStack . . .

Programmierung

Shell, C, Perl und Python haben bisher stets zur Lösung geführt

Untested software is broken by design!

Versionskontrolle

evolution

- sccs, 1972 von Marc J. Rochkind, Bell Labs
- RCS, Anfang der 1980er, Purdue University
- CVS, ca. 1994, cvs.savannah.gnu.org
- Apache subversion, . . . mercurial, . . . bazaar, . . . bitkeeper, . . .
- Lizenzänderung bei bitkeeper, April 2005, Linus Torvalds erschafft git
- git entstand zunächst zur Verwaltung der Linux-Quellen

git – was ist anders

- kein zentraler Server notwendig
- hohe Sicherheit gegenüber versehentlichem Löschen oder Verfälschen
- Viele Beteiligte können eine Sache mit vielen Versionen machen
- git (zu deutsch: Blödmann) ist einfach und gut

git init

git init

man git-init

```
NAME
       git - the stupid content tracker

SYNOPSIS
       git [--version] [--help] [-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>]

DESCRIPTION
       Git is a fast, scalable, distributed revision control system with an unusually rich
       command set that provides both high-level operations and full access to internals.
...

EXAMPLES
           Start a new git repository for an existing code base

           $ cd /path/to/my/codebase
           $ git init      (1)
           $ git add .     (2)

           1. prepare /path/to/my/codebase/.git directory
           2. add all existing file to the index
```


git init

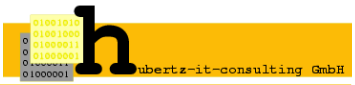
introspection

```
hans@jhx:~$ mkdir git-examples/                                                    1
hans@jhx:~$                                                                        2
hans@jhx:~$ cd git-examples/                                                       3
hans@jhx:~/git-examples$                                                           4
hans@jhx:~/git-examples$ ls -la                                                    5
insgesamt 0                                                                        6
drwxr-xr-x   2 hans hans  4096 Apr 21 21:26 .                                      7
drwxr-xr-x 185 hans hans 24576 Apr 21 21:29 ..                                     8
hans@jhx:~/git-examples$                                                           9
hans@jhx:~/git-examples$                                                           10
hans@jhx:~/git-examples$ git init                                                  11
Initialized empty Git repository in /home/hans/git-examples/.git/                  12
hans@jhx:~/git-examples$                                                           13
hans@jhx:~/git-examples$ ls -la                                                    14
insgesamt 32                                                                       15
drwxr-xr-x   3 hans hans  4096 Apr 21 21:33 .                                      16
drwxr-xr-x 185 hans hans 24576 Apr 21 21:32 ..                                     17
drwxr-xr-x   7 hans hans  4096 Apr 21 21:33 .git                                   18
hans@jhx:~/git-examples$                                                           19
```

git init

introspection

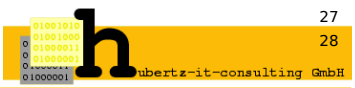
```
hans@jhx:~/git-examples$ ls -la .git/
insgesamt 40
drwxr-xr-x 7 hans hans 4096 Apr 21 21:35 .
drwxr-xr-x 3 hans hans 4096 Apr 21 21:35 ..
drwxr-xr-x 2 hans hans 4096 Apr 21 21:35 branches
-rw-r--r-- 1 hans hans   92 Apr 21 21:35 config
-rw-r--r-- 1 hans hans   73 Apr 21 21:35 description
-rw-r--r-- 1 hans hans   23 Apr 21 21:35 HEAD
drwxr-xr-x 2 hans hans 4096 Apr 21 21:35 hooks
drwxr-xr-x 2 hans hans 4096 Apr 21 21:35 info
drwxr-xr-x 4 hans hans 4096 Apr 21 21:35 objects
drwxr-xr-x 4 hans hans 4096 Apr 21 21:35 refs

hans@jhx:~/git-examples$ cat .git/description
Unnamed repository; edit this file 'description' to name the repository.

hans@jhx:~/git-examples$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```



git add

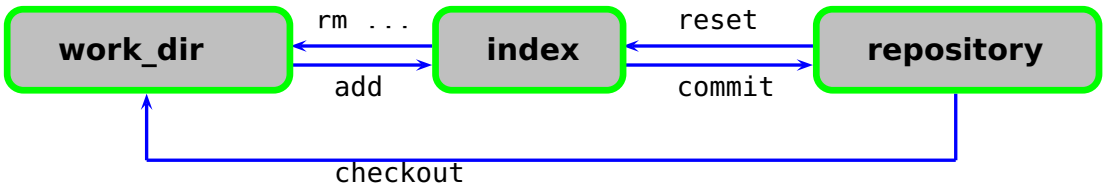
howto start

```
hans@jhx:~/git-examples$ vi first.py
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ cat first.py
#!/usr/bin/env python

print "Hello_World!"
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   first.py
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git add first.py
hans@jhx:~/git-examples$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   first.py
#
```



workflow commands

git workflow commands



stage: `git add file`

unstage: `git reset file`

checkout: `git checkout file`

commit: `git commit file`

git commit

git commit

man git-commit

```
GIT-COMMIT(1)                          Git Manual                          GIT-COMMIT(1)
NAME
       git-commit - Record changes to the repository
SYNOPSIS
       git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
                  [--dry-run] [(-c | -C | --fixup | --squash) <commit>]
                  [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
                  [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
                  [--date=<date>] [--cleanup=<mode>] [--status | --no-status]
                  [-i | -o] [--] [<file>...]

DESCRIPTION

       Stores the current contents of the index in a new commit along with a log message from
       the user describing the changes.
       The content to be added can be specified in several ways:
        1. by using git add to incrementally "add" changes to the index before using the commit
           command (Note: even modified files must be "added");
        2. by using git rm to remove files from the working tree and the index, again before
           using the commit command;
        3. by listing files as arguments to the commit command, in which case the commit will
           ignore changes staged in the index, and instead record the current content of the
           listed files (which must already be known to git);
        4. by using the -a switch with the commit command to automatically "add" changes from
           all known files (i.e. all files that are already listed in the index) and to
           automatically "rm" files in the index that have been removed from the working tree,
           and then perform the actual commit;
        5. by using the --interactive or --patch switches with the commit command to decide one
           by one which files or hunks should be part of the commit, before finalizing the
           operation. See the  Interactive Mode  section of git-add(1) to learn how to operate
           these modes.
```

commit example

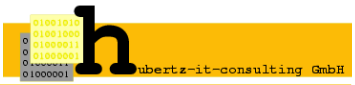
```
hans@jhx:~/git-examples$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   first.py
#
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git commit first.py -m'our very first git commit'
[master (root-commit) ac5969f] our very first git commit
 1 file changed, 3 insertions(+)
  create mode 100755 first.py
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git status
# On branch master
nothing to commit (working directory clean)
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git log
commit ac5969fb122155f5b5d3a1f57856badfd18d2dc2
Author: sl0 <sl0.self@googlemail.com>
Date:   Thu May 2 19:10:40 2013 +0200

    our very first git commit
hans@jhx:~/git-examples$
```


git config

git config

man git-config

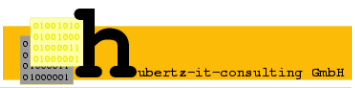
```
GIT-CONFIG(1)                    Git Manual                    GIT-CONFIG(1)

NAME
       git-config - Get and set repository or global options

SYNOPSIS
       git config [<file-option>] [type] [-z|--null] name [value [value_regex]]
       git config [<file-option>] [type] --add name value
       git config [<file-option>] [type] --replace-all name value [value_regex]
       git config [<file-option>] [type] [-z|--null] --get name [value_regex]
       git config [<file-option>] [type] [-z|--null] --get-all name [value_regex]
       git config [<file-option>] [type] [-z|--null] --get-regexp name_regex [value_regex]
       git config [<file-option>] --unset name [value_regex]
       git config [<file-option>] --unset-all name [value_regex]
       git config [<file-option>] --rename-section old_name new_name
       git config [<file-option>] --remove-section name
       git config [<file-option>] [-z|--null] -l | --list
       git config [<file-option>] --get-color name [default]
       git config [<file-option>] --get-colorbool name [stdout-is-tty]
       git config [<file-option>] -e | --edit

DESCRIPTION
       You can query/set/replace/unset options with this command. The name is actually the
       section and the key separated by a dot, and the value will be escaped.

       Multiple lines can be added to an option by using the --add option. If you want to
       update or unset an option which can occur on multiple lines, a POSIX regexp value_regex
       needs to be given. Only the existing values that match the regexp are updated or unset.
       If you want to handle the lines that do not match the regex, just prepend a single
       exclamation mark in front (see also the section called EXAMPLES).

       ...
```



git config

config example

```
hans@jhx:~/git-examples$                                    45
hans@jhx:~/git-examples$ git config --local --list          46
core.repositoryformatversion=0                              47
core.filemode=true                                          48
core.bare=false                                             49
core.logallrefupdates=true                                  50
hans@jhx:~/git-examples$                                    51
hans@jhx:~/git-examples$                                    52
hans@jhx:~/git-examples$ git config --global --list         53
user.name=sl0                                               54
user.email=sl0.self@googlemail.com                          55
color.ui=auto                                               56
color.diff=auto                                             57
color.status=auto                                           58
merge.tool=vimdiff                                          59
push.default=current                                        60
hans@jhx:~/git-examples$                                    61
```



.gitignore

.gitignore

Manche Dinge sollen nicht in die Versionskontrolle

.gitignore hilft mit regulären Ausdrücken:

```
*.aux
*.log
*.out
*.pdf
*.toc
pics/*
```

workflow in a

multiuser environment

common multiuser commands

git branch: list, create, remove,

git checkout: set workspace to a specific commit or branch

git diff: show difference from a to b

git stash: list, pop, rm

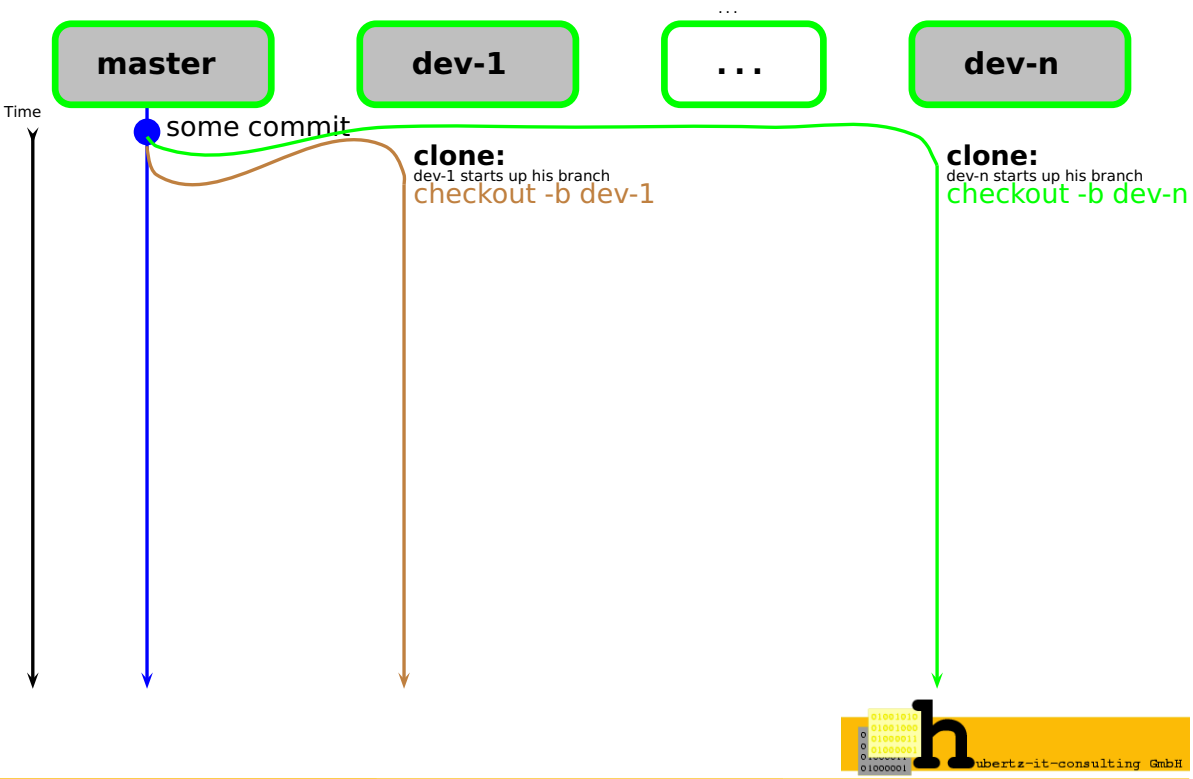
git merge: integrate another branch into current

common multiuser commands: **branch**

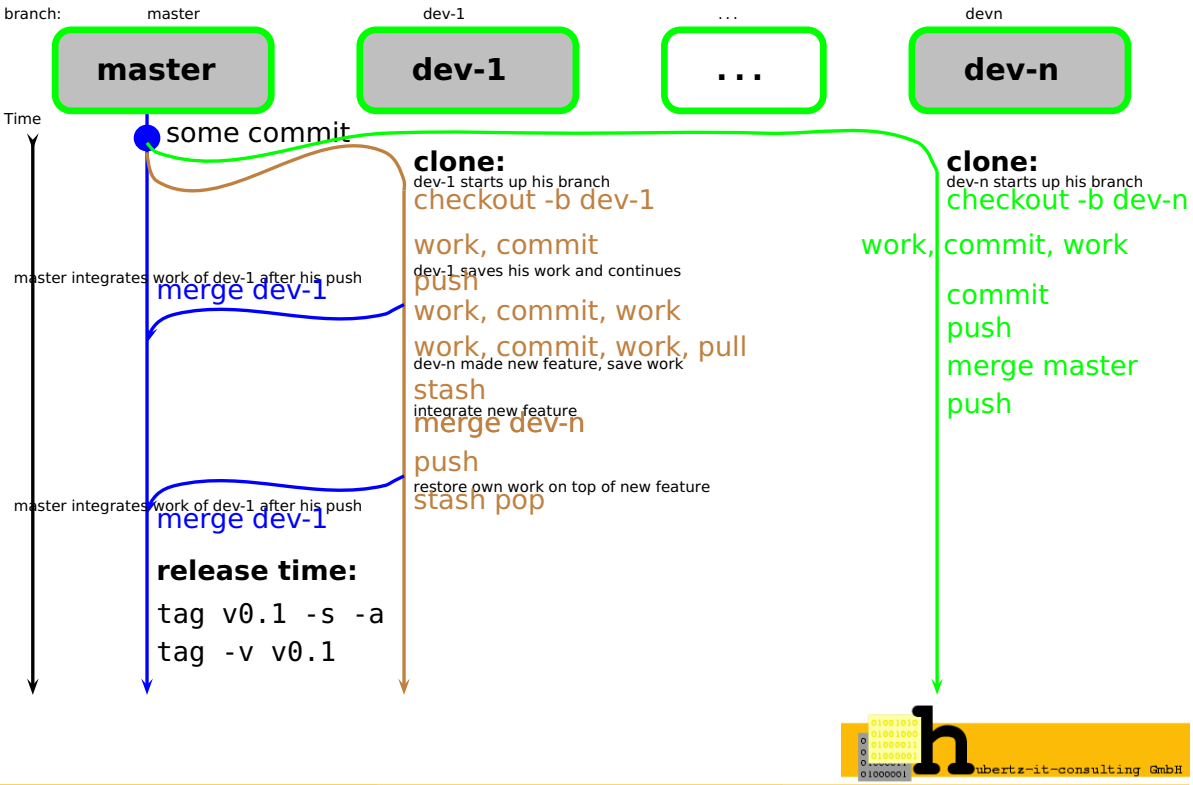
```
NAME
       git-branch - List, create, or delete branches

SYNOPSIS
       git branch [--color[=<when>] | --no-color] [-r | -a]
               [--list] [-v [--abbrev=<length> | --no-abbrev]]
               [(--merged | --no-merged | --contains) [<commit>]] [<pattern>...]
       git branch [--set-upstream | --track | --no-track] [-l] [-f] <branchname>
               [<start-point>]
       git branch (-m | -M) [<oldbranch>] <newbranch>
       git branch (-d | -D) [-r] <branchname>...
       git branch --edit-description [<branchname>]

DESCRIPTION
       With no arguments, existing branches are listed and the current branch
       will be highlighted with an asterisk. Option -r causes the
       remote-tracking branches to be listed, and option -a shows both. This
       list mode is also activated by the --list option (see below).
       <pattern> restricts the output to matching branches, the pattern is a
       shell wildcard (i.e., matched using fnmatch(3)). Multiple patterns
       may be given; if any of them matches, the branch is shown.
```


git multiuser workflow

git multiuser workflow



git multiuser workflow



git merge

git merge

man git-merge

```
NAME
       git-merge - Join two or more development histories together
SYNOPSIS
       git merge [-n] [--stat] [--no-commit] [--squash] [--[no-]edit]
               [-s <strategy>] [-X <strategy-option>]
               [--[no-]rerere-autoupdate] [-m <msg>] [<commit>...]
       git merge <msg> HEAD <commit>...
       git merge --abort
DESCRIPTION
       ...
       Assume the following history exists and the current branch is "master":

                     A---B---C topic
                    /
               D---E---F---G master

       Then "git merge topic" will replay the changes made on the topic branch since it
       diverged from master (i.e., E) until its current commit (C) on top of master, and
       record the result in a new commit along with the names of the two parent commits
       and a log message from the user describing the changes.

                     A---B---C topic
                    /         \
               D---E---F---G---H master

       ...
       Warning: Running git merge with uncommitted changes is discouraged: while possible,
       it leaves you in a state that is hard to back out of in the case of a conflict.
       ...
```

git merge: fast-forward or conflict

git merge command shows, whats happening: **fast-forward** or **conflict**

- Merge result is one out of two:
  1) Merge succeded, no overlapping changes detected
  2) Merge conflicts, overlapping changes found
- how to continue in case of success:

  finalize:

  git commit
- how to continue in case of conflicts:

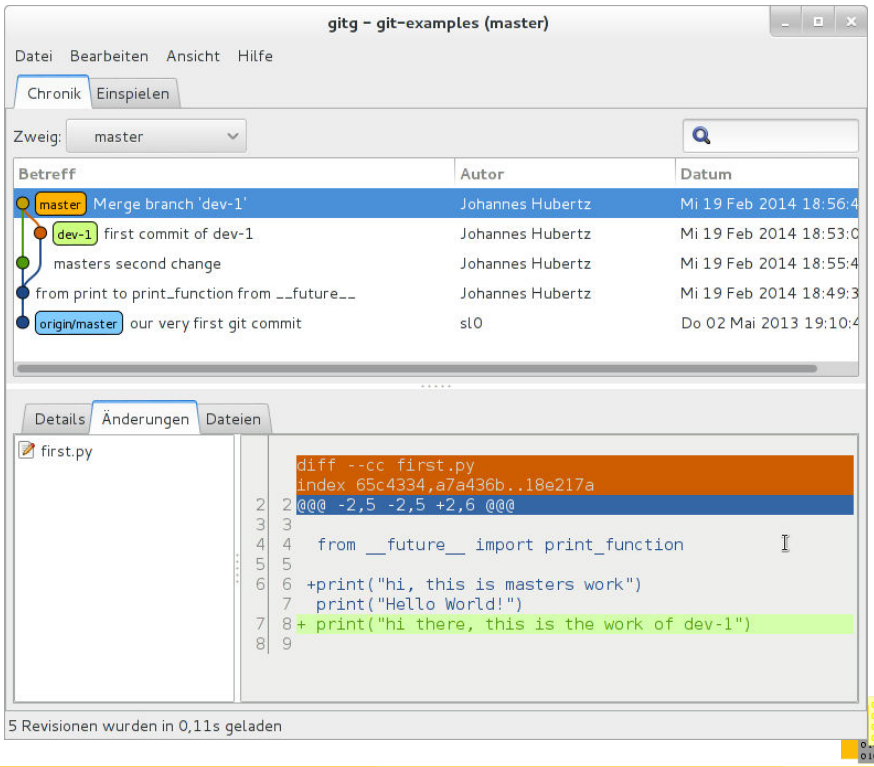
  keep cool, it's really hard to break a git!
  resolve conflicts through editing:

  git mergetool

  finalize:

  git commit

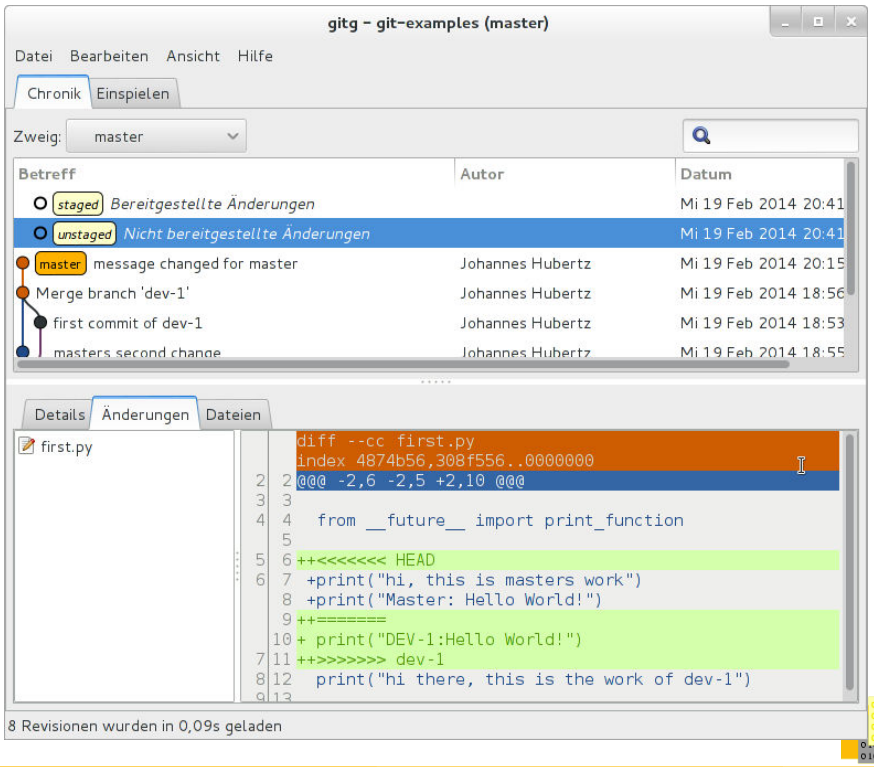
gitg easy going . . .



git merge with conflict

```
hans@jhx:~/git-examples$ git merge dev-1
Auto-merging first.py
CONFLICT (content): Merge conflict in first.py
Automatic merge failed; fix conflicts and then commit the result.
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ cat first.py
#!/usr/bin/env python

from __future__ import print_function

<<<<<<< HEAD
print("hi,_this_is_masters_work")
print("Master:_Hello_World!")
=======
print("DEV-1:Hello_World!")
>>>>>>> dev-1
print("hi_there,_this_is_the_work_of_dev-1")
hans@jhx:~/git-examples$   #### look at lines 13, 16 and 18 ####
hans@jhx ~/git-examples$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#both modified:      first.py
#
no  changes added to commit (use "git_add" and/or "git_commit_-a")
hans@jhx ~/git-examples$
```


gitg accidentially not easy going . . .



git merge with conflict

```
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:      first.py

no changes added to commit (use "git add" and/or "git commit -a")
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$
```


git mergetool

git mergetool

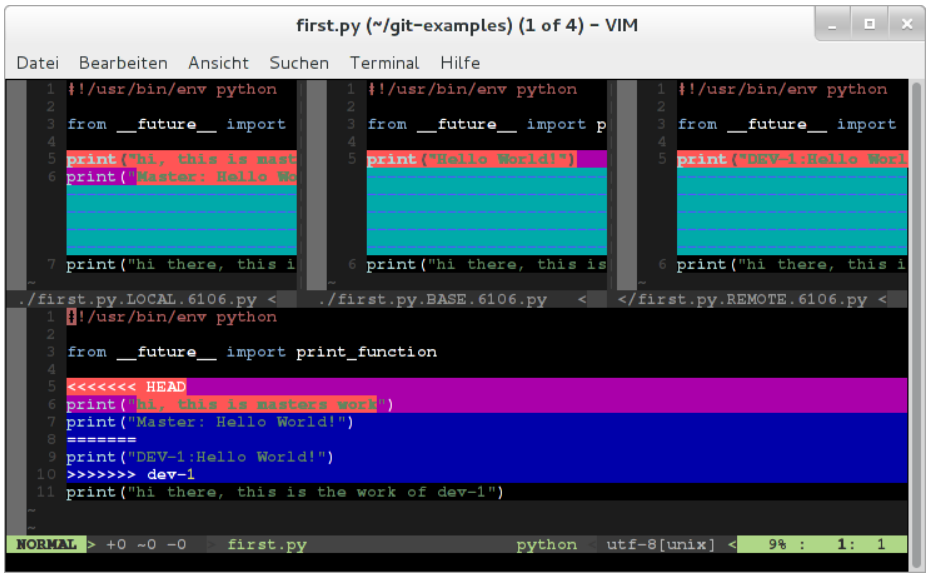
`man git-mergetool`

```
NAME
       git-mergetool - Run merge conflict resolution tools to resolve merge conflicts
SYNOPSIS
       git mergetool [--tool=<tool>] [-y|--no-prompt|--prompt] [<file>...]
DESCRIPTION
       Use git mergetool to run one of several merge utilities to resolve merge conflicts.
       It is typically run after git merge.

       If one or more <file> parameters are given, the merge tool program will be run to
       resolve differences on each file (skipping those without conflicts). Specifying a
       directory will include all unresolved files in that path. If no <file> names are
       specified, git mergetool will run the merge tool program on every file with merge
       conflicts.
OPTIONS
       -t <tool>, --tool=<tool>
           Use the merge resolution program specified by <tool>. Valid merge tools are:
           araxis, bc3, diffuse, ecmerge, emerge, gvimdiff, kdiff3, meld, opendiff,
           p4merge, tkdiff, tortoisemerge, vimdiff and xxdiff.

           If a merge resolution program is not specified, git mergetool will use the
           configuration variable merge.tool. If the configuration variable merge.tool
           is not set, git mergetool will pick a suitable default.

           You can explicitly provide a full path to the tool by setting the
           configuration variable mergetool.<tool>.path. For example, you can configure
           the absolute path to kdiff3 by setting mergetool.kdiff3.path. Otherwise, git
           mergetool assumes the tool is available in PATH.
           ...
```

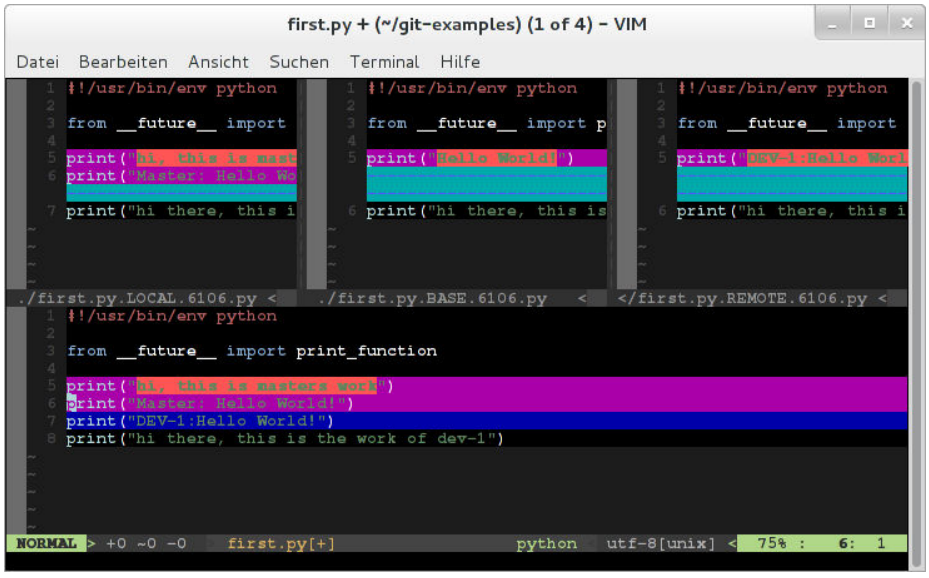

git merge with conflict

```
hans@jhx:~/git-examples$ git mergetool
Merging:
first.py

Normal merge conflict for 'first.py':
{local}: modified file
{remote}: modified file
Hit return to start merge resolution tool (vimdiff):
```

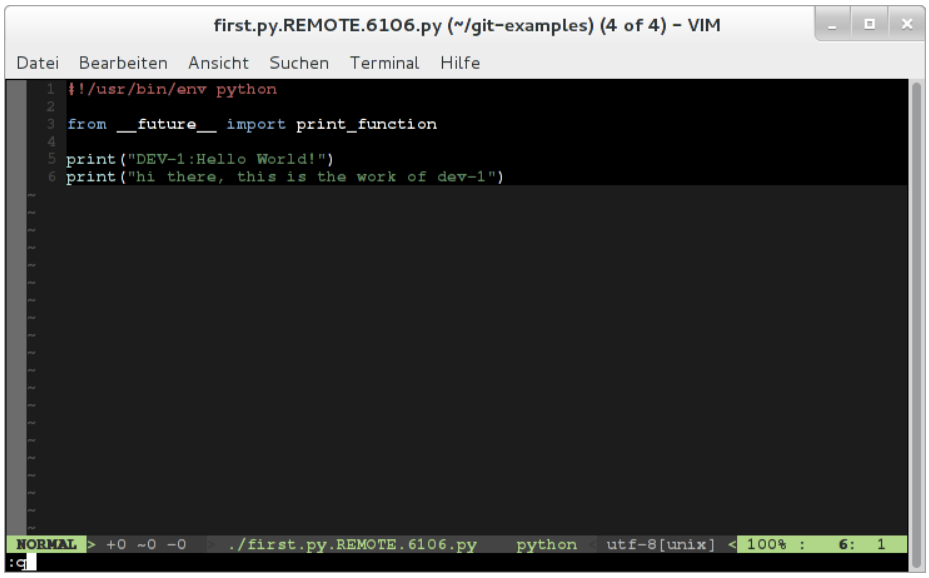
git mergetool startup from commandline . . .



git mergetool editing finished . . .



git mergetool closing all the partials of vim . . .



git merge resolved: merge conflict finalization

```
hans@jha ~/git-examples$ ls -l
insgesamt 8
-rwxr-xr-x 1 hans hans 199 Feb 26 20:08 first.py
-rwxr-xr-x 1 hans hans 234 Feb 26 20:04 first.py.orig
hans@jha ~/git-examples$
hans@jhx:~/git-examples$ git status
# On branch master
# Changes to be committed:
#
#modified:   first.py
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#first.py.orig
hans@jhx:~/git-examples$
hans@jhx:~/git-examples$ git commit first.py -m'conflict removed, all OK' -i
[master aabfebf] conflict removed, all OK
hans@jhx:~/git-examples$
```

missing git commands

there are more and more and more commands for git

- git show
- git shortlog
- git blame
- git branch
- git tag
- git remote
- git clone
- git fetch
- git pull
- git push
- git reset
- git rebase
- git gui
- gitg, gitk
- gitweb, gitolite
- and more and more and more . . .

Quellen und Lesestoff

```
Erstlektüre:
apt-get install git ; man gittutorial
http://www.deimeke.net/dirk/blog/index.php?/archives/3298-Einfuehrung-in-Git-....html

Übung:
http://githowto.com/

Selbermachen:
https://github.com/         Lizenzbedingungen beachten!

Bücher:
Preißel, Stachmann: GIT, dpunkt.verlag, 2012
Haenel, Plenz: Git, OpenSourcePress, 2011
Sven Riedel: Git kurz & gut, O'Reilly, 2009
Chacon: Pro Git, APress, 2009, online: http://git-scm.com/book

Links zur Quelle:
https://git.kernel.org/cgit/git/git.git/
https://git.kernel.org/cgit/git/git-manpages.git/
https://git.kernel.org/cgit/git/git-htmldocs.git/
```


Ich bedanke mich für Ihre Aufmerksamkeit

hubertz-it-consulting GmbH jederzeit zu Ihren Diensten:

verlässliche Netzwerke für vertrauliche Kommunikation
Ihre Sicherheit ist uns wichtig!

Frohes Schaffen

Johannes Hubertz

it-consulting _at_ hubertz dot de

powered by LaTeX 2ε and PSTricks