Karlsruhe Institute of Technology

Distributed version control and why *you* want to use it

Matthias Vogelgesang

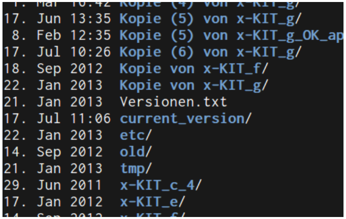
Institute for Data Processing and Electronics

Plead guilty!

It’s easy to copy digital content, so why not re-create it over and over again?

Plead guilty!

It’s easy to copy digital content, so why not re-create it over and over again?

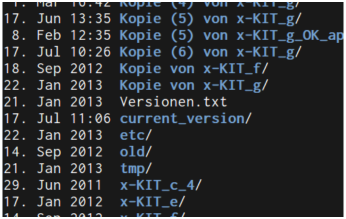


```
 1. Mar 10:42 Kopie (4) von x-KIT_g/
17. Jun 13:35 Kopie (5) von x-KIT_g/
 8. Feb 12:35 Kopie (5) von x-KIT_g_OK_ap
17. Jul 10:26 Kopie (6) von x-KIT_g/
18. Sep 2012  Kopie von x-KIT_f/
22. Jan 2013  Kopie von x-KIT_g/
21. Jan 2013  Versionen.txt
17. Jul 11:06 current_version/
22. Jan 2013  etc/
14. Sep 2012  old/
21. Jan 2013  tmp/
29. Jun 2011  x-KIT_c_4/
17. Jan 2012  x-KIT_e/
14. Sep 2012  x-KIT_f/
```

“One of these folders *must* contain the latest version …”

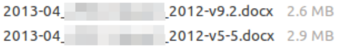
Plead guilty!

It’s easy to copy digital content, so why not re-create it over and over again?



```
17. Jun 13:35 Kopie (5) von x-KIT_g/
 8. Feb 12:35 Kopie (5) von x-KIT_g_OK_ap
17. Jul 10:26 Kopie (6) von x-KIT_g/
18. Sep 2012  Kopie von x-KIT_f/
22. Jan 2013  Kopie von x-KIT_g/
21. Jan 2013  Versionen.txt
17. Jul 11:06 current_version/
22. Jan 2013  etc/
14. Sep 2012  old/
21. Jan 2013  tmp/
29. Jun 2011  x-KIT_c_4/
17. Jan 2012  x-KIT_e/
```

“One of these folders *must* contain the latest version …”



```
2013-04_[redacted]_2012-v9.2.docx  2.6 MB
2013-04_[redacted]_2012-v5-5.docx  2.9 MB
```

“Here is the latest version of the proposal/paper/report.” — “Thanks.”

Obvious disadvantages

- No meta data about *what* was changed *when* by *whom*
- You lose track of what’s going on
- You cannot roll-back to a working state
- Poor solution for collaboration

Version control

Benefits

- *Track* files
- Record (*commit*) changes
- Share changes with others
- Roll-back to an earlier state

Centralized version control systems

Implementations

- File-based: RCS
- Client-Server architecture: CVS, SVN, …

Centralized version control systems

Implementations

- File-based: RCS
- Client-Server architecture: CVS, SVN, …

Problems

- Centralized systems require a server
- Interaction with a repository can be painfully slow
- Setup and maintenance issues
- Collaborating requires a lot of effort

Distributed version control

Cloned repositories

- Local setup
- Blazingly fast operations
- “Airplane coding”

Sharing is an inherent feature

- DVCS are built around the idea of sharing
- Cryptographic hashing of content assures integrity
- Easy branching and merging of changes between peers

Distributed version control *systems*

Mercurial, Bazaar, SVK, Monotone, BitKeeper, **Git**, Darcs, Fossil, GNU arch, Arx, Plastic SCM

Why Git?

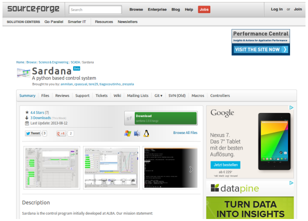
Karlsruhe Institute of Technology

Pros

- De-facto standard for open source software
- Probably the fastest DVCS out there
- GitHub has more sex appeal than sf.net

Cons

- Command line interface can be a bit inconsistent
- Git is a toolbox with much freedom and little limits



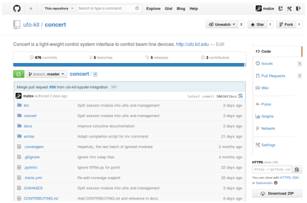
Sardana

A python based control system

Brought to you by: anmilan, cpascual, tere29, tiagocoutinho, zreszela

Description

Sardana is the control program initially developed at ALBA. Our mission statement:



ufo-kit / concert

Concert is a light-weight control system interface to control beam line devices. http://ufo.kit.edu — Edit

676 commits   5 branches   5 releases   3 contributors

concert / +

Merge pull request #98 from ufo-kit/spyder-integration

matze authored 2 days ago   latest commit 50636f1bcc

bin	Split session module into utils and management	2 days ago
concert	Split session module into utils and management	2 days ago
docs	Improve coroutine documentation	2 days ago
extras	Adapt completion script for mv command	21 days ago
.coveragerc	Hopefully, the last batch of ignored modules	2 months ago
.gitignore	Ignore Vim swap files	4 months ago
.pylintrc	Ignore tifffile.py for pylint	22 days ago
.travis.yml	Re-add coverage support	22 days ago
CHANGES	Split session module into utils and management	2 days ago
CONTRIBUTING.rst	Add CONTRIBUTING.rst and reference in docs	6 days ago

You can clone with HTTPS, SSH, or Subversion.

Git basics

Getting started

Installation

- Debian/Ubuntu: `apt-get install git-core`
- openSUSE/SLES: `zypper install git-core`
- Fedora/RHEL/CentOS/SL: `yum install git`
- Mac: `port install git-core` or install from `http://git-scm.com/download/mac`
- Windows: install from `http://git-scm.com/download/win`

Creating a new repository

In the working directory of your project, type

```
$ git init
```

Tracking files

```
$ git add <path>
```

Committing changes

```
$ git commit
```

Excursion: File states

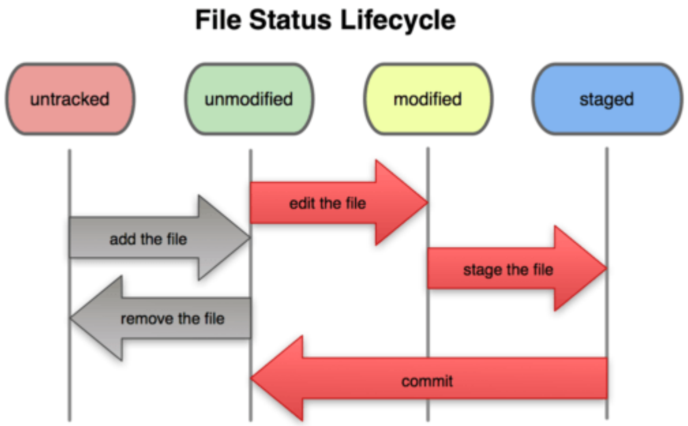


Figure: from Scott Schacon's “Pro Git” CC-BY-NC-SA 3.0

Checking the status

```
$ git status
```

Staging changes

So, before committing you have to stage a file

```
$ vi paper.tex
$ git add paper.tex
$ git commit
```

or in one go

```
$ git commit -a
```

Visualizing the history

For a quick look

```
$ git log
```

GUIs

- gitg
- gitk
- giggle
- tig
- ...

Branches

To explore an idea without messing with the original work, you can create a branch off of it …

```
$ git branch fancy-idea
$ git checkout fancy-idea
```

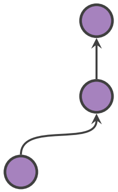
Branches

To explore an idea without messing with the original work, you can create a branch off of it …

```
$ git branch fancy-idea
$ git checkout fancy-idea
```

and commit changes related to that idea

```
$ git commit …
```



Branches

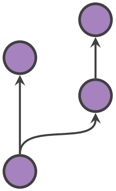
To explore an idea without messing with the original work, you can create a branch off of it …

```
$ git branch fancy-idea
$ git checkout fancy-idea
```

and commit changes related to that idea

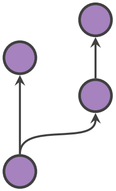
```
$ git commit …
```

Branches are cheap, so don't bother creating as many as you like.



Merging changes

If your changes are ready for prime time, merge them into your master branch:

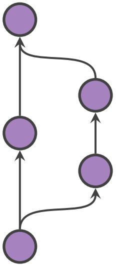


Merging changes

If your changes are ready for prime time, merge them into your master branch:

```
$ git checkout master
$ git merge fancy-idea
```

In this particular case, a *merge commit* will be created.



Merging changes

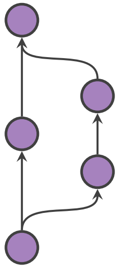
If your changes are ready for prime time, merge them into your master branch:

```
$ git checkout master
$ git merge fancy-idea
```

In this particular case, a *merge commit* will be created.

If merging was successful, the old branch can be removed

```
$ git branch -D fancy-idea
```



Collaborating with others

- Until now, everything happened on our local machine
- To share changes with others you can
  - Send patches
  - *pull* from a remote repository
  - *push* from a remote repository
- Remotes does not have to be a single server instance
- Different workflows can be easily modeled

Remote repositories

Cloning repositories

```
$ git clone {file,ssh,https}://foo.server.com/foo.git
```

Adding additional repositories

```
$ git remote add foo https://foo.server.com/foo.git
```

Syncing changes

```
$ git pull [<remote> <branch>]
$ git push [<remote> <local-branch>:<remote-branch>]
```

Hosting repositories

- Some directory on a file share such as NFS
- Simple SSH-based access or Gitolite
- Third-party provider, e.g. GitHub, Bitbucket, Google Code, SourceForge…

Best practices

- Write descriptive commit messages and keep 50/70 limits
- Check the status before committing
- Think twice before running

```
$ git commit -a
```

Advanced Git operations

Manipulating the Git object database

If you collaborate heavily with your peers, you'll want to have a "clean" history of changes, e.g.

- Concise commit messages
- One commit per logical change
- A series of commits leading to a bigger change

Fixing the last commit

Change author and message

```
$ git commit --amend
```

Picking cherries

Pull individual commits into a branch

```
$ git cherry-pick f023bac
```

Partial staging

Staging only relevant parts of a change

```
git add -p/--patch
```

Stash intermediate changes away

Cleaning the working directory temporarily

```
$ git stash "Descriptive message"
$ ... do something else
$ git stash pop
```

Merge bubbles

Merging branches that developed independently can end up nasty …

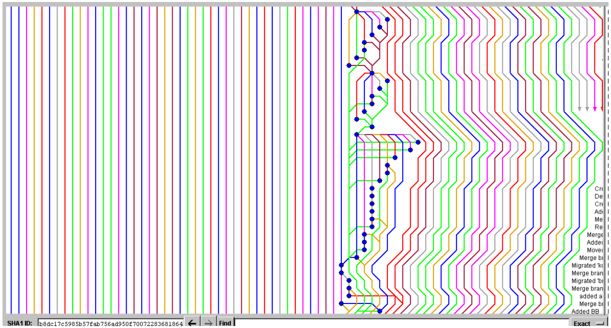


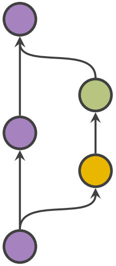
Figure: “Successful” octopus merge.

Image from: `http://blog.spearce.org/2007/07/difficult-gitk-graphs.html`

Rebasing branches

This can be reduced by rebasing one branch on top of the other

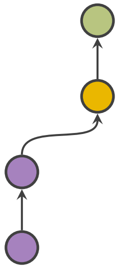
```
$ git checkout some-feature
```



Rebasing branches

This can be reduced by rebasing one branch on top of the other

```
$ git checkout some-feature
$ git rebase master
```

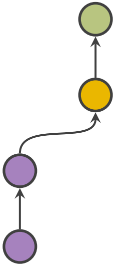


Rebasing branches

This can be reduced by rebasing one branch on top of the other

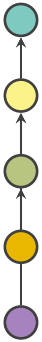
```
$ git checkout some-feature
$ git rebase master
```

No merge commit, clean history.



Re-writing the history

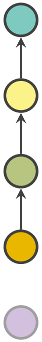
Manipulate the change history by rebasing using the `-i`/`--interactive` switch



Re-writing the history

Manipulate the change history by rebasing using the `-i/--interactive` switch

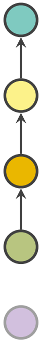
- Drop commits



Re-writing the history

Manipulate the change history by rebasing using the `-i/--interactive` switch

- Drop commits
- Re-order commits



Re-writing the history

Manipulate the change history by rebasing using the `-i`/`--interactive` switch

- Drop commits
- Re-order commits
- Squash several commits into one

Re-writing the history

Manipulate the change history by rebasing using the `-i`/`--interactive` switch

- Drop commits
- Re-order commits
- Squash several commits into one
- Edit commits

Re-writing the history

Manipulate the change history by rebasing using the `-i`/`--interactive` switch

- Drop commits
- Re-order commits
- Squash several commits into one
- Edit commits

```
$ git rebase -i HEAD~4
```

Best practices

- Keep a clean history by re-writing the history of your local branch
- Never, ever re-write the history of a public branch (Once pushed, a change is sacred)

Beyond version control

Deployment, Continous Integration

post-receive hooks

what	Use post-receive hooks to trigger actions, e.g. running builds and tests, deploy software, …
where	`$REPO/.git/hooks`
score	● ○ ○ ○ ○

Blogging

GitHub pages + Jekyll

what	Blog hosting on GitHub via Git and Jekyll
where	`pages.github.com`
score	●●○○○

Wiki

Gollum + Smeagol

what	Git backend for a wiki with Markdown formatting
where	`github.com/gollum/gollum` and `github.com/rubyworks/smeagol`
score	●●○○○

Backups

Bup

what	Uses Git’s packfile format to store backups
where	`github.com/bup/bup`
score	●●○○○

Managing large files

git-annex

what	Manages large data sets across remotes
where	`git-annex.branchable.com`
score	●●●●○

Bug tracking

ticgit

what	Keep tickets in a separate branch and sync across repos
where	`github.com/jeffWelling/ticgit`
score	● ● ● ● ○

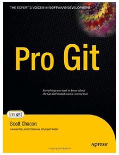
Text-based slides

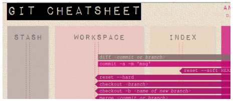
git-slides

what	git-slides (together with Vim)
where	`github.com/gelisam/git-slides`
score	

Further reads

- `$ man git` ... just kidding
- Free Pro Git book at `git-scm.com/book`
- Different aspects from beginners to pros: `gitready.com`
- Git cheat sheet: `ndpsoftware.com/git-cheatsheet.html`
- Interactive walkthrough: `gitimmersion.com`





Thanks for your attention!

Get these slides from: `github.com/matze/kseta-dvsc-talk`
Title logo by Jason Long licensed under CC-BY-3.0