

+++ git

git

the fast version control system



Git Tutorial

Bastian Venthur
venthur@cs.tu-berlin.de

Berlin Institute of Technology
2009-05

Outline

- Introduction to git
 - What is git
 - Features of git
- Tutorial
 - Standalone developer
 - Participating developer
 - git-svn
- If time permits: Demo

Part I: Introduction to Git

What is Git?

- A free, *distributed* revision control system
- Initially created by Linus Torvalds in 2005
- Runs on all major platforms
- (*British, slang*): A silly, incompetent, stupid, annoying or childish person.

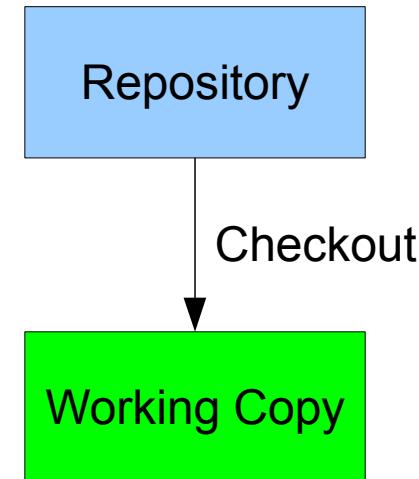
I'm an egotistical bastard, and I name all my projects after myself.
First Linux, now git.



Distributed vs Centralized

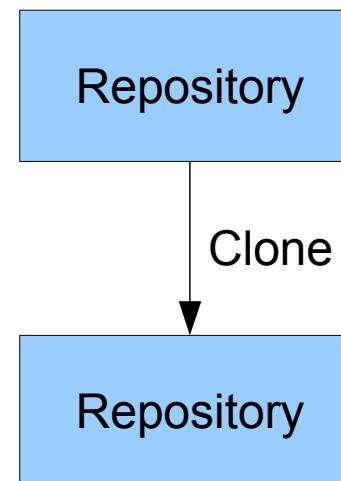
Centralized

- You *checkout* a *working copy* (the tip of the source)



Distributed

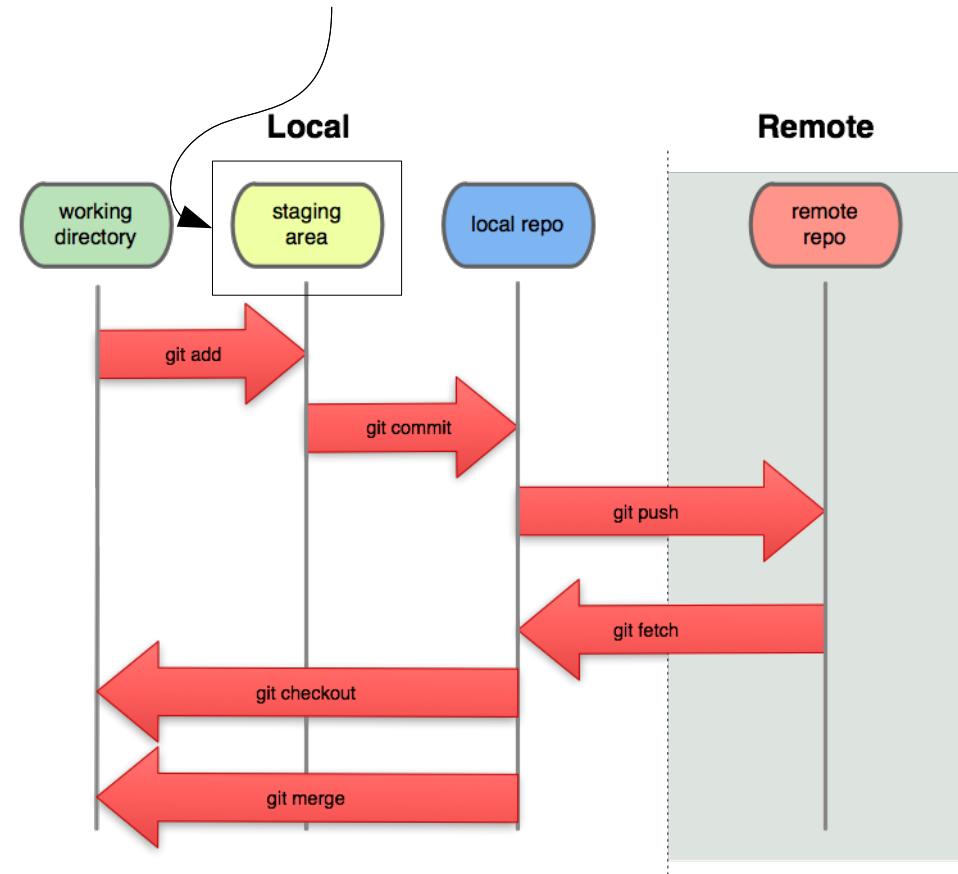
- You *clone* the repository (a full backup)



Implication (I): Everything is local

Very little outside “push”, (Just ignore that for a moment.)
“pull” and “fetch”
communicates with
anything else than your
hard disk

- Most operations are much faster
- Allows you to work offline



Implication (II): Instant revision control

Putting your paper/thesis/talk/etc. under revision control is just a **git init** away!

```
someproject
|-- file1
|-- file2
|-- file3
|-- somesubdir1
|   |-- filebar
|   '-- filefoo
 '-- somesubdir2
    '-- file
```

git init

```
someproject
|-- .git
|-- file1
|-- file2
|-- file3
|-- somesubdir1
|   |-- filebar
|   '-- filefoo
 '-- somesubdir2
    '-- file
```

repository

working copy

Git's Features

- Fast
- Small
- The staging area
- Cheap local branching
- Any workflow



Git is fast

	Git	Hg	Bzr
Init	0.024s	0.059s	0.600s
Add	8.535s	0.368s	2.381s
Status	0.451s	1.946s	14.744s
Diff	0.543s	2.189s	14.248s
Tag	0.056s	1.201s	1.892s
Log	0.711s	2.650s	9.055s
Commit (Large)	12.480s	12.500s	23.002s
Commit (Small)	0.086s	0.517s	1.139s
Branch (Cold)	1.161s	94.681s	82.249s
Branch (Hot)	0.070s	12.300s	39.411s

“Add” with over 2000 files!

First and second time
a repository was branched.

The figure consists of eight bar charts arranged in two rows of four. Each chart compares the execution times of three version control systems: git (blue), hg (light blue), and bzr (dark blue). The operations are: Init, Add, Status, Diff, Tag, Log, Commit (Lg), and Commit (Sm). In all cases, git shows significantly faster performance than hg and bzr, particularly for 'Add' and 'Commit (Large)' which involve large file sets. The 'Branch (Hot)' operation shows a significant performance difference between the first and second time it is performed, with hg showing a massive increase in time for the second branch.

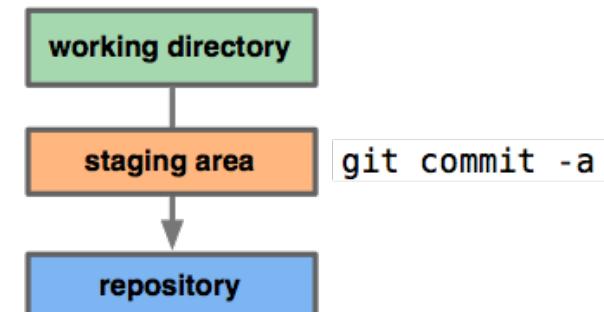
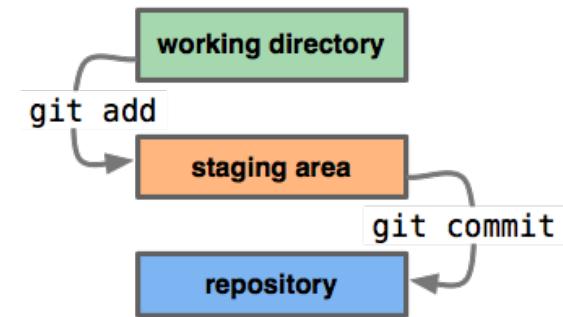
Operation	git	hg	bzr
Init	~0.024	~0.059	~0.600
Add	~8.535	~0.368	~2.381
Status	~0.451	~1.946	~14.744
Diff	~0.543	~2.189	~14.248
Tag	~0.056	~1.201	~1.892
Log	~0.711	~2.650	~9.055
Commit (Large)	~12.480	~12.500	~23.002
Commit (Small)	~0.086	~0.517	~1.139
Branch (Cold)	~1.161	~94.681	~82.249
Branch (Hot)	~0.070	~12.300	~39.411

Git is small

- Git stores it's data very efficiently
- The .git directory of the current Linux kernel (containing 4 years of the 2.6 line) is only half of the size than a checkout

The staging area

- Intermediate area to store changes before committing them
- Why is this cool?
 - Make two logically unrelated changes
 - Stage the first file and commit it
 - Stage the second file and commit it
 - Works even on a patch basis within one file!
- But you don't have to use it:



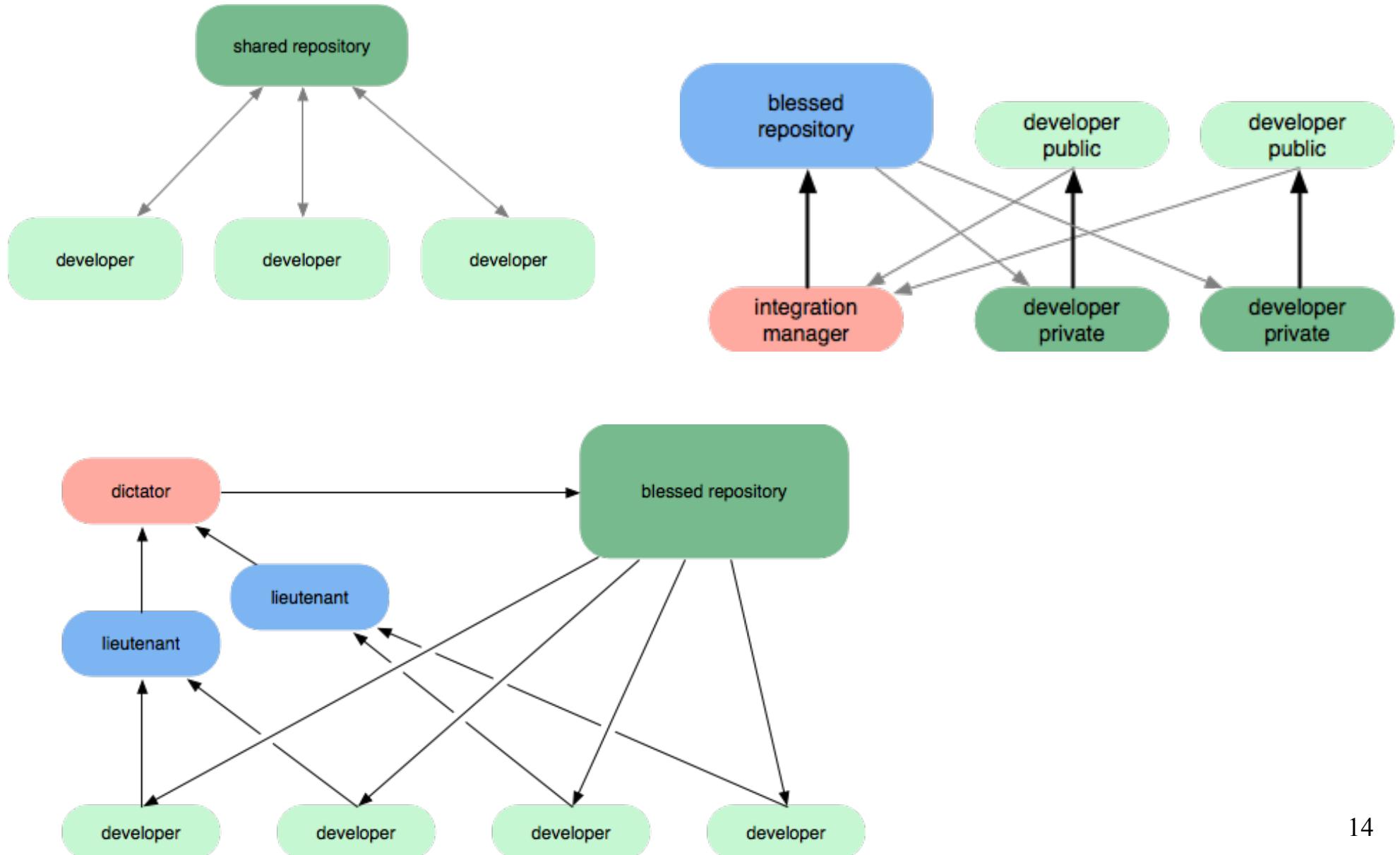
Question

Does everyone know what a branch is?

Cheap local branching

- Git's most compelling feature
- Creating, merging and deleting branches is fast, cheap and easy!
- Switching back and forth between branches is seamless
- Git really encourages you to create new branches, try stuff out – merge branches, etc

Any workflow



Who is using Git?

- Linux kernel
- X.org Server
- Qt
- GNOME
- Samba
- Perl
- VLC
- Wine
- Ruby on Rails
- rsync
- Android mobile platform
- ...



ANDROID

Questions so far?

Part II: Tutorial

Standalone developer

- **git init** to create a new repository
- **git add** to manage the index file.
- **git commit** to advance the current branch.
- **git diff** and **git status** to see what you are in the middle of doing.
- **git checkout** and **git branch** to switch branches.
- **git merge** to merge between local branches.
- **git log** to see what happened.
- **git show-branch** to see where you are.
- **git reset** and **git checkout** (with pathname parameters) to undo changes.
- **git tag** to mark known point.
- **git rebase** to maintain topic branches.

Creating a new repository

\$ git init

Creates an empty git repository

\$ git add .

A snapshot of the current directory is now stored in a temporary staging area called the index.

\$ git commit

Stores the first version of the project in the repository

Making changes

\$ (edit file1, file2, file3)

\$ git diff

See what changed between your working copy and the index

\$ git add file1 file2 file3

Add the updated contents to the index

\$ git status

Get a brief overview of the situation

\$ git commit

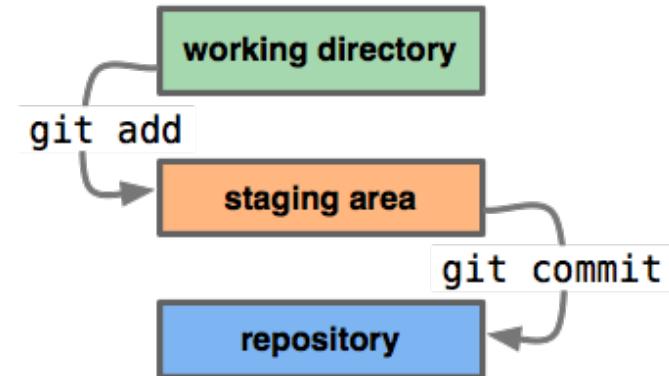
Commit the changes to the repository

Making changes (II)

```
$ (edit file1, file2, file3)
```

```
$ git add file1 file2 file3
```

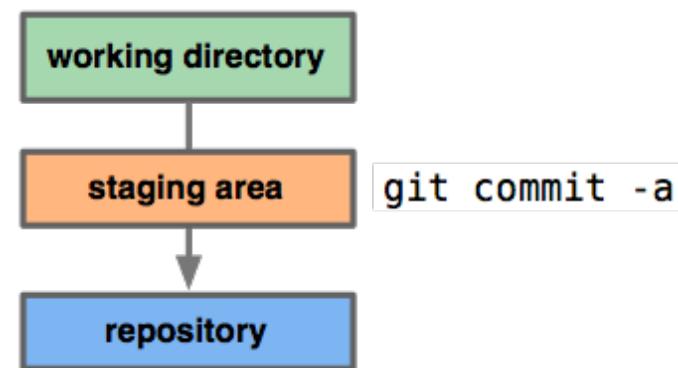
```
$ git commit
```



Is equivalent to:

```
$ (edit file1, file2, file3)
```

```
$ git commit -a
```



Making changes (III)

You can even:

```
$ (edit file1, file2, file3)
$ git add file1
$ git commit -m "nifty new feature"
$ git add file2 file3
$ git commit -m "bugfix"
```

And even:

```
$ git add --interactive
```

Managing branches

\$ git branch experimental

Creates a new branch called “experimental”

\$ git branch

experimental

*** master**

Shows where we are

\$ git checkout experimental

Switches to branch “experimental”

\$ git branch

*** experimental**

master

Managing branches (II)

See the difference between two branches:

\$ (edit files in experimental)

\$ git commit -a

\$ git diff master

Merging two branches:

\$ git checkout master

\$ git merge experimental

Without conflicts you're done here, else:

\$ (resolve conflicts)

\$ git commit -a

Removing a branch:

\$ git branch -d experimental

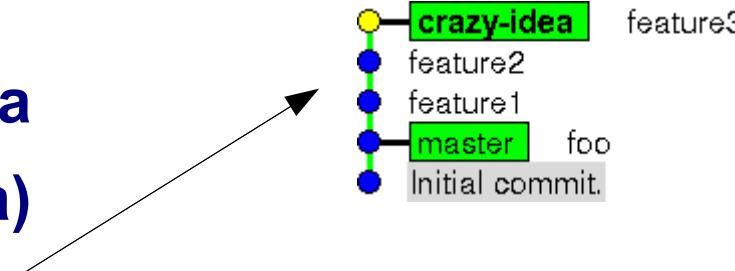
Managing branches (III)

```
$ git branch crazy-idea
```

```
$ git checkout crazy-idea
```

```
$ (edit files in crazy-idea)
```

```
$ git commit -a
```

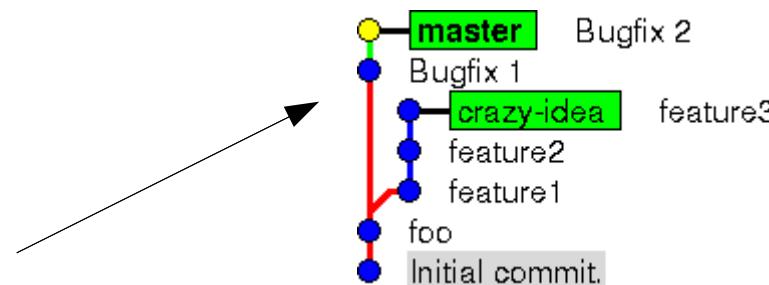


```
$ git checkout master
```

```
$ (fix a bug in master)
```

```
$ git checkout crazy idea
```

```
$ git merge master
```



```
$ git branch -D crazy-idea
```



Reverting changes

\$ git checkout <file>

Reverts one file back to its original state

What if I have a file called like a branch?

\$ git checkout -- <file>

\$ git checkout -f

Throws out all changes since the last commit.

Also, be aware that '**git revert**' is not equivalent to '**svn revert**'!
git-revert is used to reverse commits.



Exploring project history

\$ git log

```
commit ec06de4c3849c7b3d91b60ff09e10f1108c23cea
Author: user-a <venthur@debian.org>
Date:   Thu May 14 11:58:00 2009 +0200

    added file z
```

```
commit 301ee9f80df142e67b4de77225e3ed1af4b2036b
Merge: fef7261 54b87da
Author: user-b <venthur@debian.org>
Date:   Mon May 11 17:32:49 2009 +0200
```

Merge branch 'devel'

\$ git log -p

Like above but with patches.

Exploring project history (II)

```
$ git show-branch
```

```
* [master] Add `git show-branch`.  
! [fixes] Introduce "reset type" flag to "git reset"  
! [mhf] Allow "+remote:local" refspec to cause --force when fetching.  
---  
+ [mhf] Allow "+remote:local" refspec to cause --force when fetching.  
+ [mhf~1] Use git-octopus when pulling more than one heads.  
+ [fixes] Introduce "reset type" flag to "git reset"  
+ [mhf~2] "git fetch --force".  
+ [mhf~3] Use .git/remote/origin, not .git/branches/origin.  
+ [mhf~4] Make "git pull" and "git fetch" default to origin  
+ [mhf~5] Infamous `octopus merge`  
+ [mhf~6] Retire git-parse-remote.  
+ [mhf~7] Multi-head fetch.  
+ [mhf~8] Start adding the $GIT_DIR/remotes/ support.  
*++ [master] Add `git show-branch`.
```

Exploring project history (III)

\$ gitk or \$ gitk --all

The screenshot shows the gitk graphical interface for a repository named 'git-test'. The interface displays a timeline of commits on the left and a detailed log on the right.

Timeline (Left):

- master branch: A yellow dot at the top.
- remotes/origin/bar: A green box containing a blue dot.
- remotes/origin/master: A green box containing a blue dot.
- remotes/origin/develop: A green box containing a blue dot.
- devel: A green box containing a blue dot.
- Added some other files.
- added some files
- remotes/origin/foo: A green box containing a blue dot.
- Merge commit 'origin/foo' into foo: A blue dot connected to the foo branch.
- beh: A blue dot connected to the foo branch.
- baz: A blue dot connected to the foo branch.
- bar: A blue dot connected to the foo branch.
- foo: A blue dot connected to the foo branch.
- some commit 3
- some commit 2
- some commit
- test-branch: A green box containing a blue dot.
- remotes/origin/test-branch: A green box containing a blue dot.

Log (Right):

Autor	Datum
Bastian Venthur <venthur@debian.org>	2009-05-27 10:12:45
Bastian Venthur <venthur@debian.org>	2009-05-27 10:12:26
user-a <venthur@debian.org>	2009-05-18 10:18:45
user-a <venthur@debian.org>	2009-05-14 11:57:15
user-a <venthur@debian.org>	2009-05-14 11:56:35
Bastian Venthur <venthur@debian.org>	2009-05-26 14:58:31
user-b <venthur@debian.org>	2009-05-26 14:57:45
user-b <venthur@debian.org>	2009-05-26 14:56:44
Bastian Venthur <venthur@debian.org>	2009-05-26 14:55:07
Bastian Venthur <venthur@debian.org>	2009-05-26 14:54:59
Bastian Venthur <venthur@debian.org>	2009-05-26 14:51:40
Bastian Venthur <venthur@debian.org>	2009-05-26 14:50:45
Bastian Venthur <venthur@debian.org>	2009-05-26 14:50:12
user-a <venthur@debian.org>	2009-05-18 10:55:37

Bottom Navigation:

- SHA1: b14a1681966093dbd9d2db3ccf7cc01661e4e443
- Suche
- nächste
- vorige
- Version nach
- Beschreibung: ▾
- Exakt
- Alle Felder

Bottom Details:

- Suche
- Vergleich
- Alte Version
- Neue Version
- Kontextzeilen: 3
- Beschreibung
- file1
- Patch
- Baum

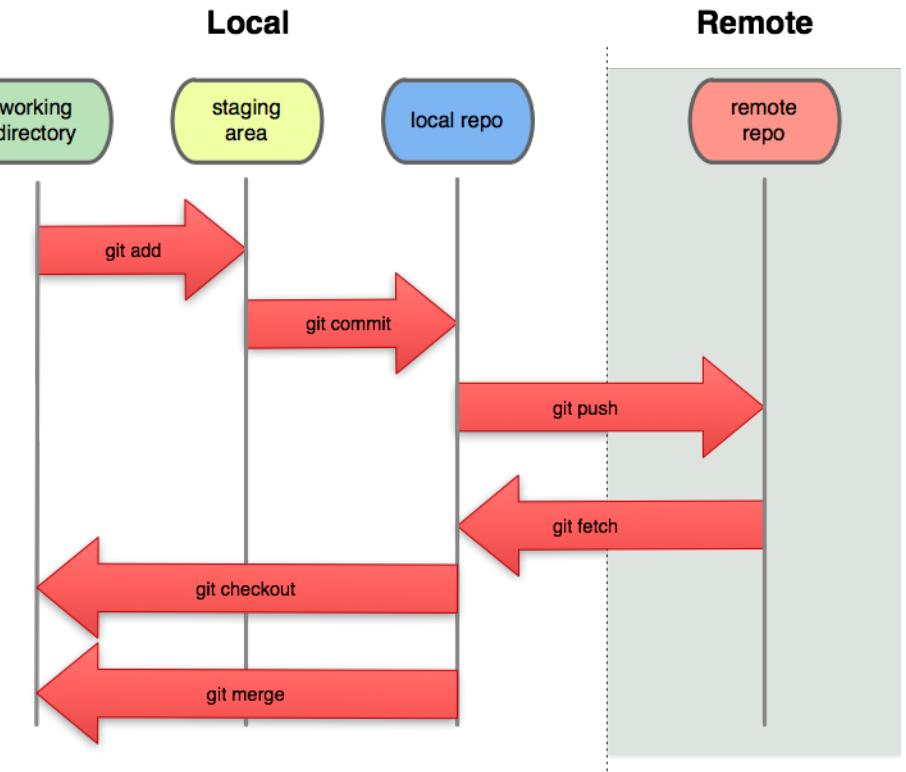
Bottom Log Summary:

- Autor: Bastian Venthur <venthur@debian.org> 2009-05-27
- Eintragender: Bastian Venthur <venthur@debian.org> 2009-05-27
- Eltern: f3f529fdde6d8bf2b8c3e0a72a132fe8fdc3be (added)
- Zweig: master, remotes/origin/bar, remotes/origin/master
- Folgt auf:
- Vorgänger von:

foo

Participating Developer

- **git clone** from the upstream to prime your local repository.
- **git pull** and **git fetch** from "origin" to keep up-to-date with the upstream.
- **git push** to shared repository, if you adopt CVS style shared repository workflow



Just to make it a *bit* more complicated...

Two kinds of branches:

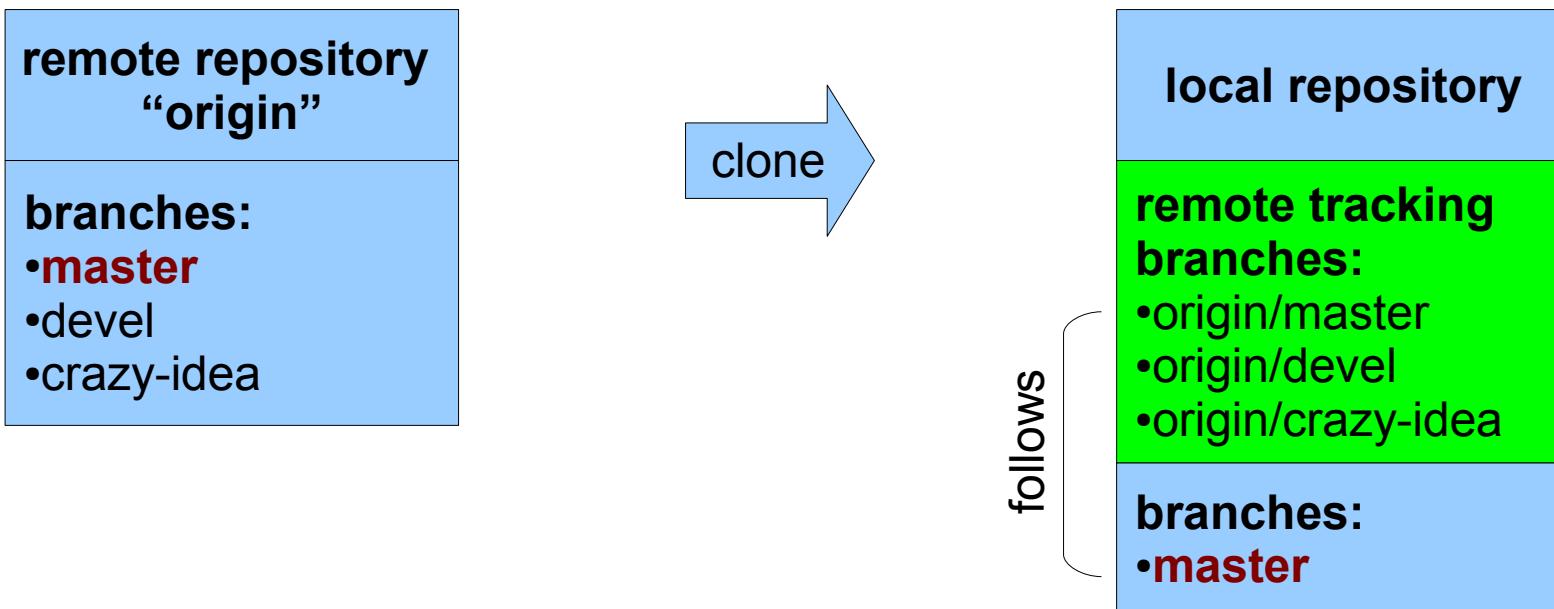
- **Local branches** (e.g.: foo)
- **Remote tracking branches** (e.g.: origin/foo)
 - Read-only
 - Follow what's happening remotely

git clone

\$ git clone git://some.re.po

- Clones repo into a newly created directory
- Creates remote tracking branches for each branch in repo
- Creates and checks out a local branch equal to repo's currently active branch

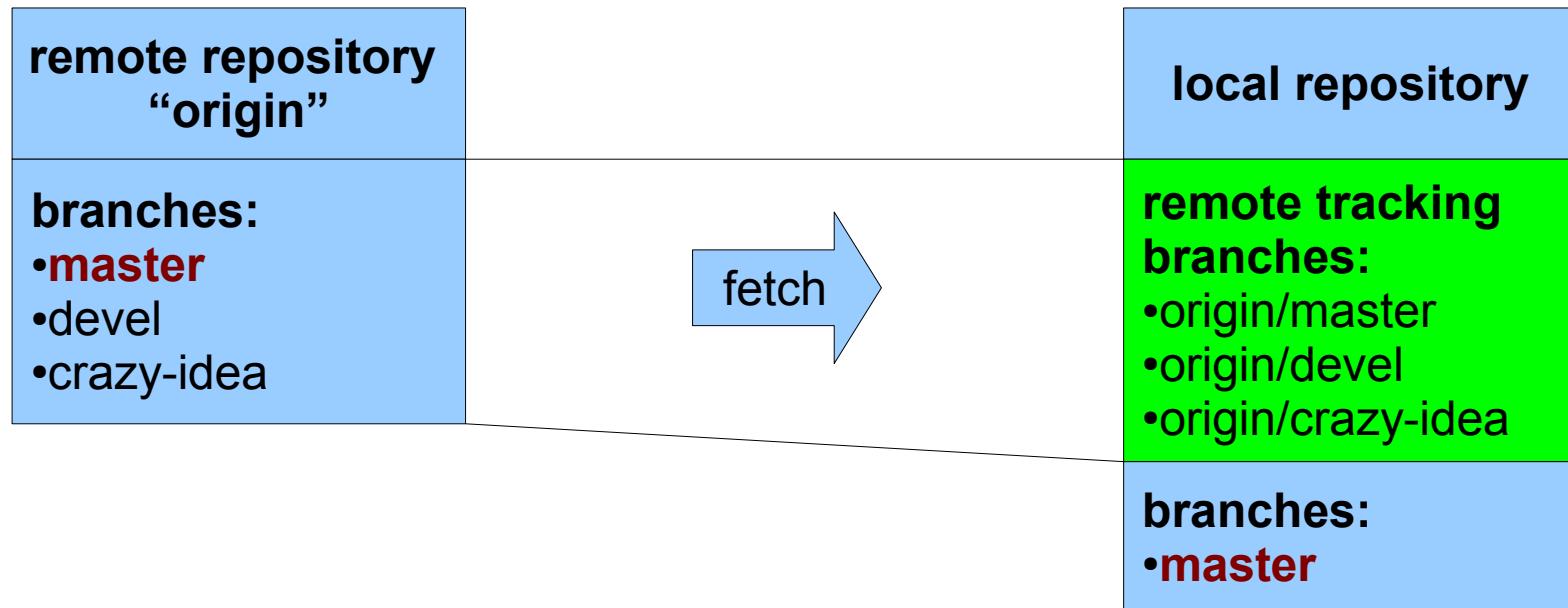
git clone (II)



git fetch

\$ git fetch

- Updates *all* remote tracking branches
- Does not touch your local branches!



git fetch (II)

- If you want to merge (update!) your local master branch with origin's changes

```
$ git checkout master
```

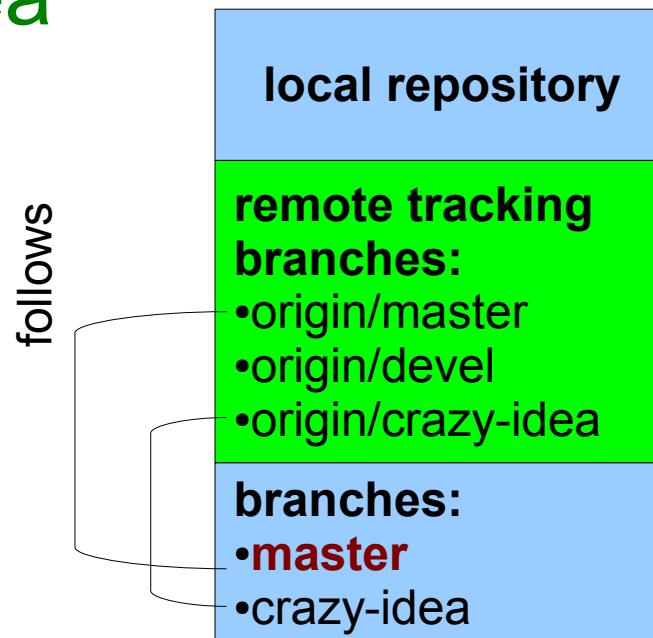
```
$ git merge origin/master
```

Creating a local branch from a remote tracking branch

Remember: remote tracking branches are read only!

```
$ git checkout --track origin/crazy-idea
```

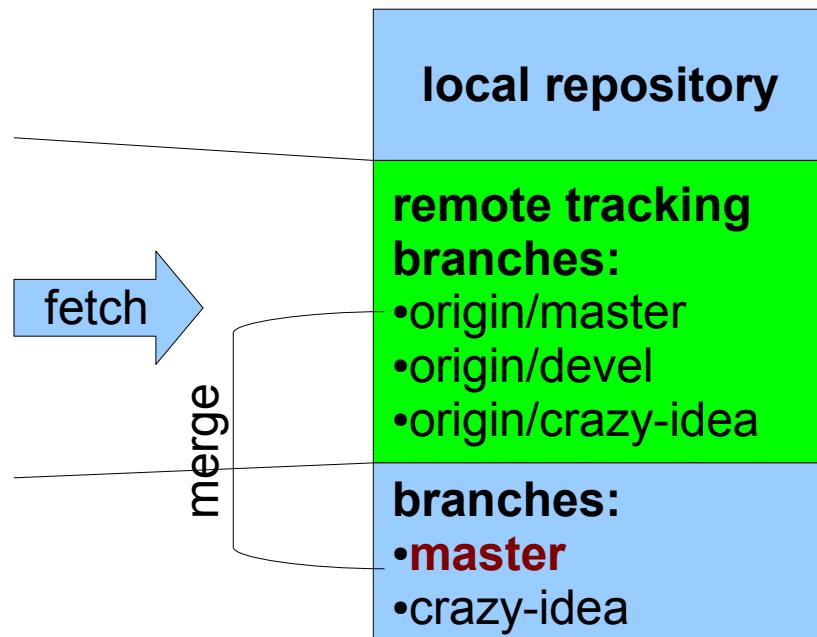
Creates a local branch **crazy-idea** which follows **origin/crazy-idea**



git pull

\$ git pull

- Updates all remote tracking branches
- Merges your current active branch the connected remote tracking branch



pull vs fetch

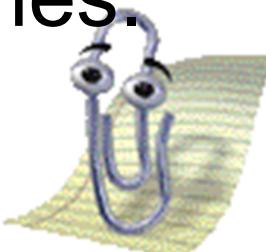
\$ git fetch

- Updates *all* remote tracking branches

\$ git pull

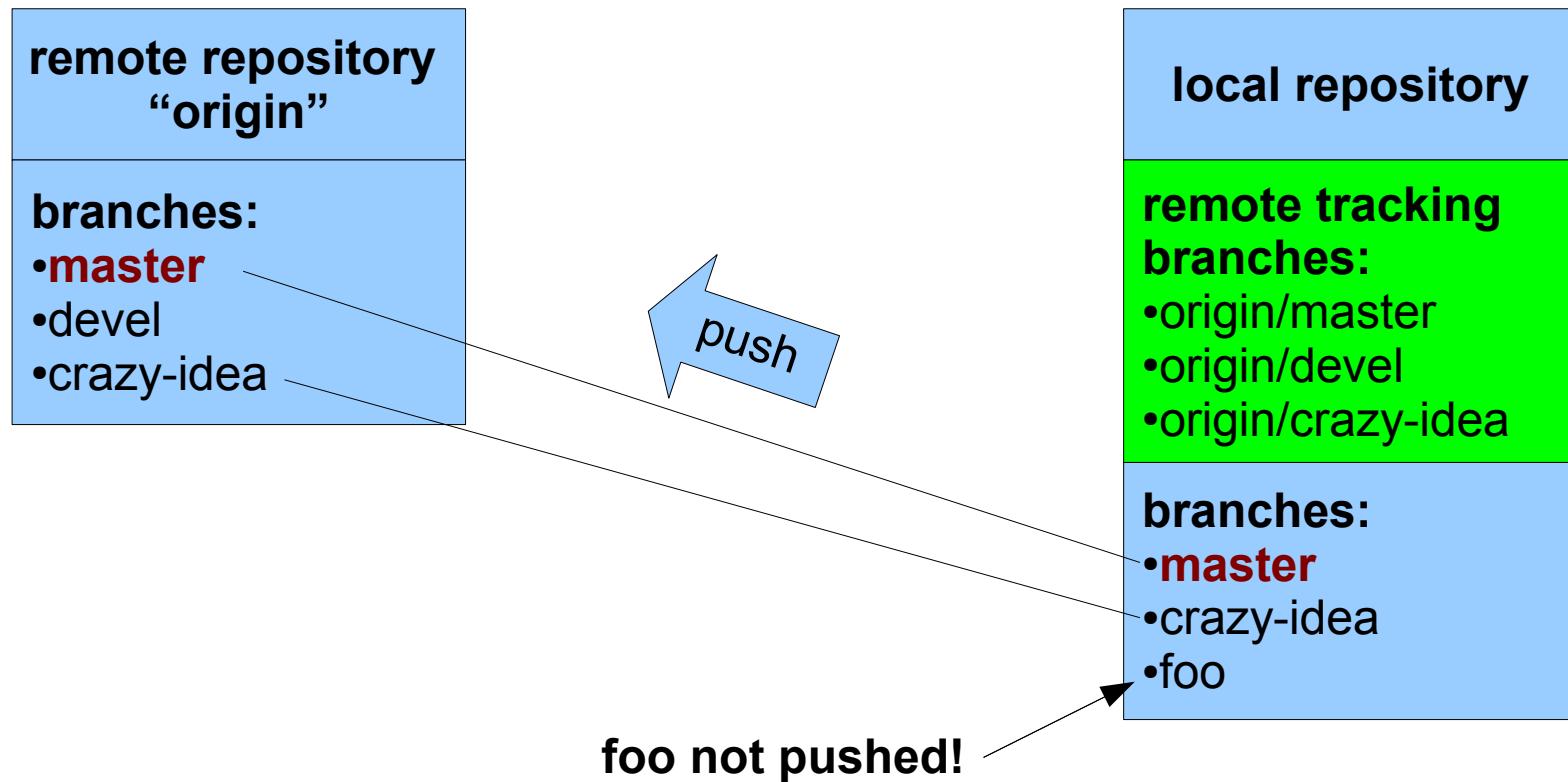
- **git fetch** for *all* remote tracking branches
- + **git merge** for the *current* branch

It's generally safe to call **git fetch** during your work, since it does not touch your local branches.



git push

Update origin's branches with all branches that are common between your local repository.



git push (II)

To push a local branch:

```
$ git push origin foo
```

- After this explicit push:
 - You have an `origin/foo` remote tracking branch “connected” with `foo`
 - every following `git push` is sufficient

Typical workflow:

Only once:

\$ git clone git://some.re.po

\$ (hacky, branch, hacky, merge, etc.)

\$ git fetch

Fetch upstream's changes

\$ git merge origin/master

Merge the changes from origin/master into current branch

\$ git push

Push your changes back to origin

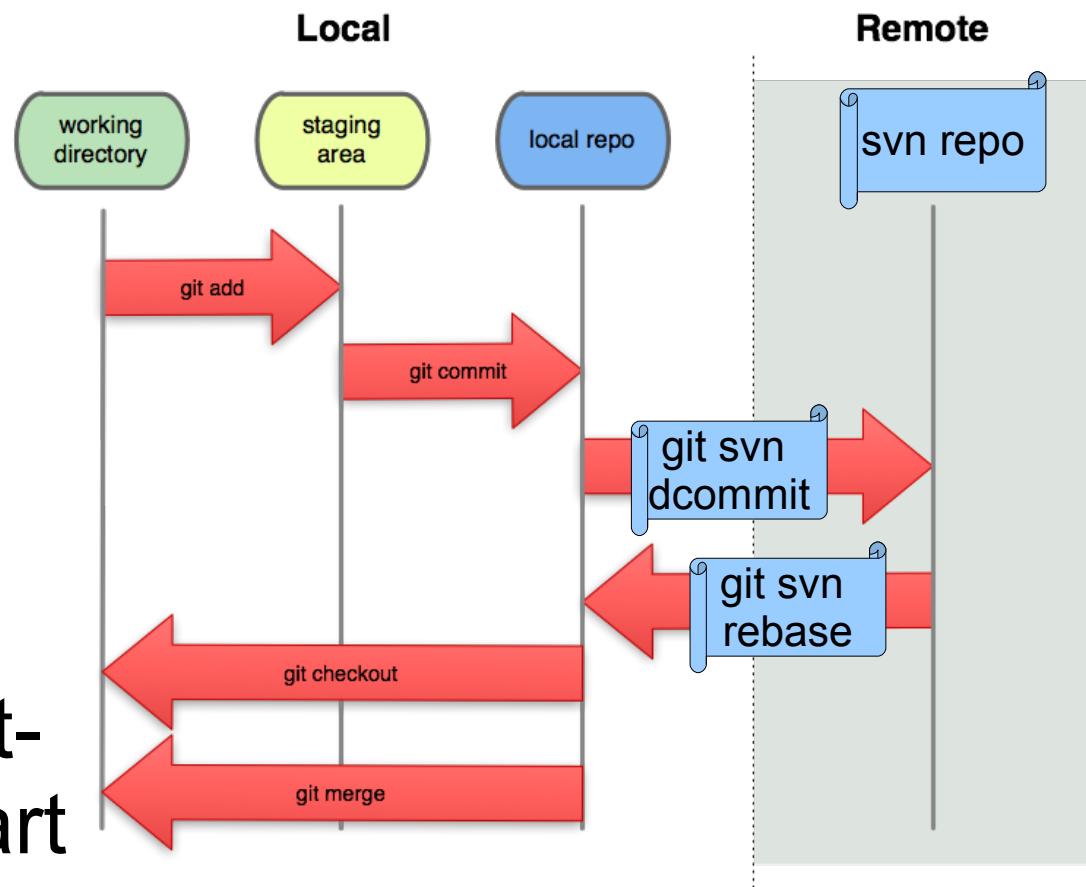
“But at work we have to use SVN!”



git-svn in a nutshell

Git SVN

- Plug-in for git
- Allows bidirectional operation between a single SVN branch and git
- Use git locally and git-svn for the remote part



git-svn in a nutshell (II)

\$ git svn clone http://svn.reposito.ry

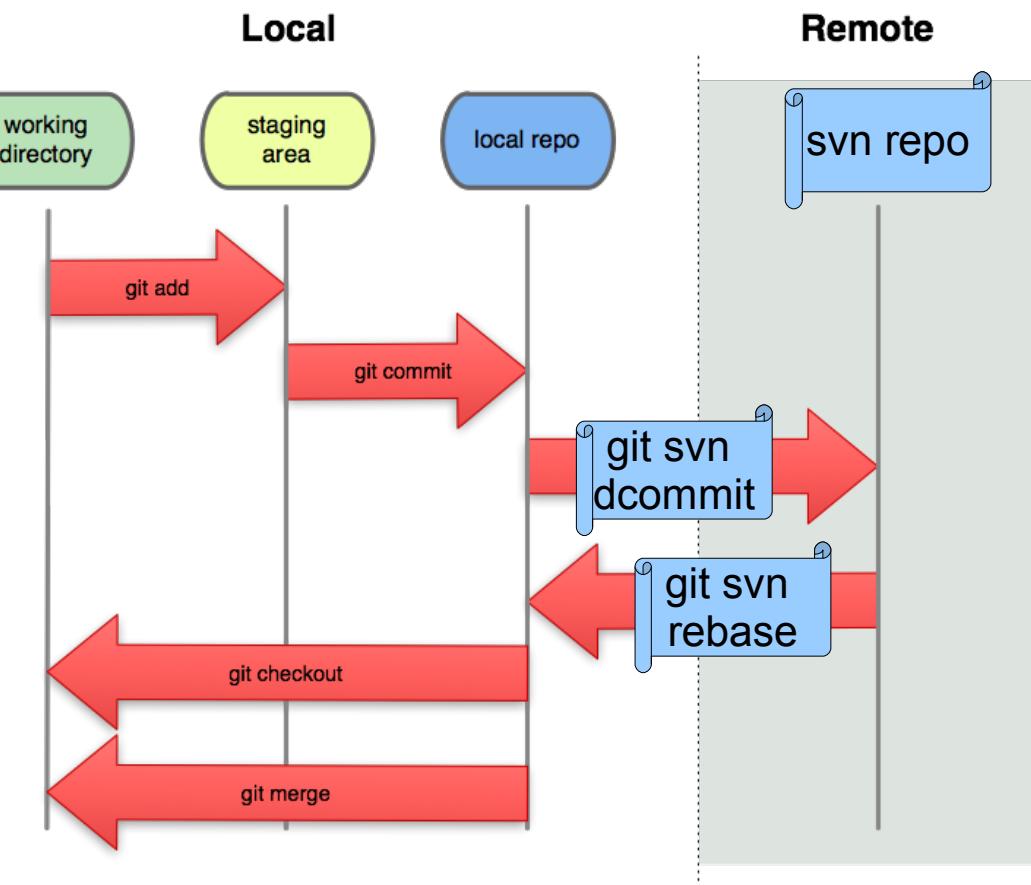
Creates empty git repository and fetches **every** revision from the svn repo into it.

\$ git svn rebase (“svn update”)

Fetches revisions from the SVN parent of the current HEAD and rebases the current (uncommitted to SVN) work against it.

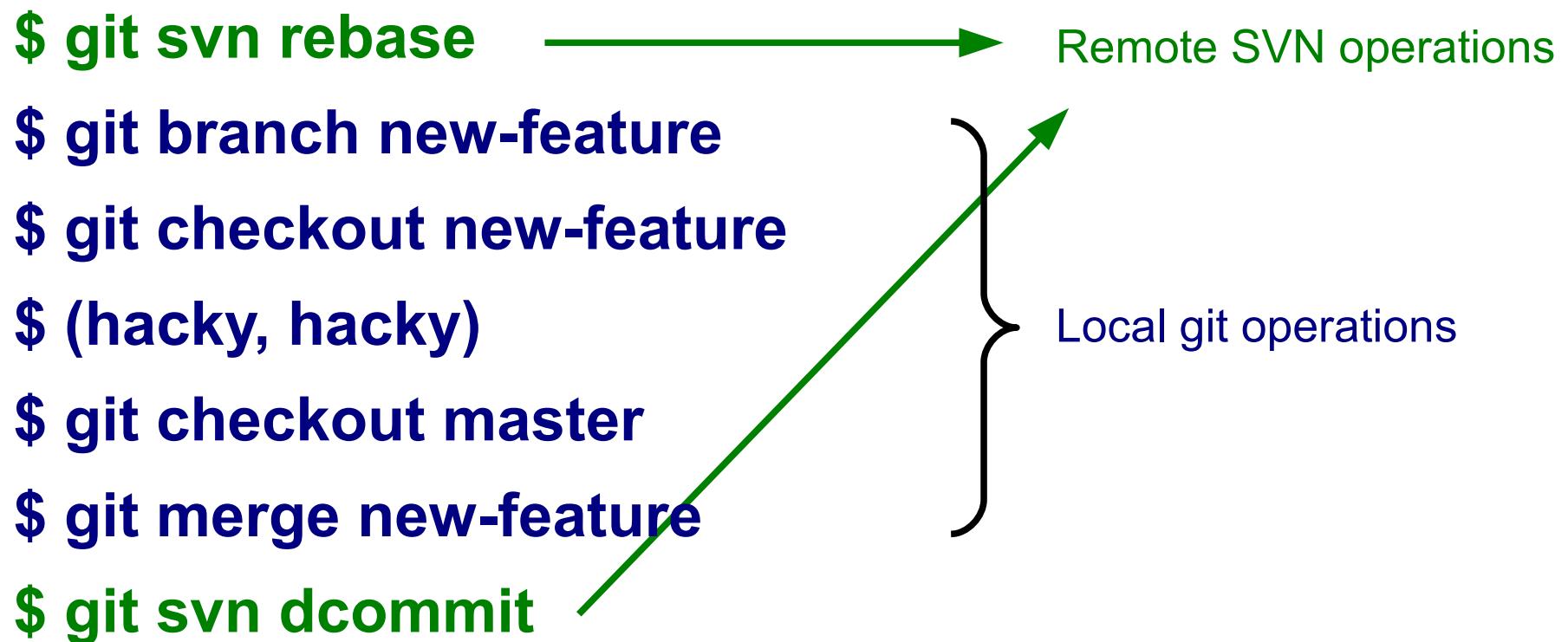
\$ git svn dcommit (“svn commit”)

Will create a revision in SVN for each commit in git.



git-svn in a nutshell (III)

- git-svn does not work well with remote (SVN) branches!
- Best to track only the main line (trunk) and use local git branches for your development



Summary

- Difference: Centralized and Distributed VCS
- Howto create/clone a repository
- Difference: local- and remote tracking branches
- Basic git usage
- Git with SVN

Next Actions:

- Take a project of your choice (thesis, paper, ...) and put it under git revision control
- Use git-svn to use git locally with SVN repositories
- Convince your group to move from SVN to git

Fin.

Useful Resources

- git tutorial:
\$ man gittutorial
- Everyday git with 20 commands or so:
<http://www.kernel.org/pub/software/scm/git/docs/everyday.html>
- git ready:
<http://www.gitready.com/>
- Why git is better than X:
<http://whygitisbetterthanx.com>
- Git Cheat Sheet:
<http://cheat.errtheblog.com/s/git>
- Git Cheat Sheet:
<http://git.or.cz/gitwiki/GitCheatSheet>
- Git community book:
<http://book.git-scm.com/index.html>