Introduction to Git

Benedikt Meurer (bm@os-cillation.de)

2012 / 05 / 11

os-cillation

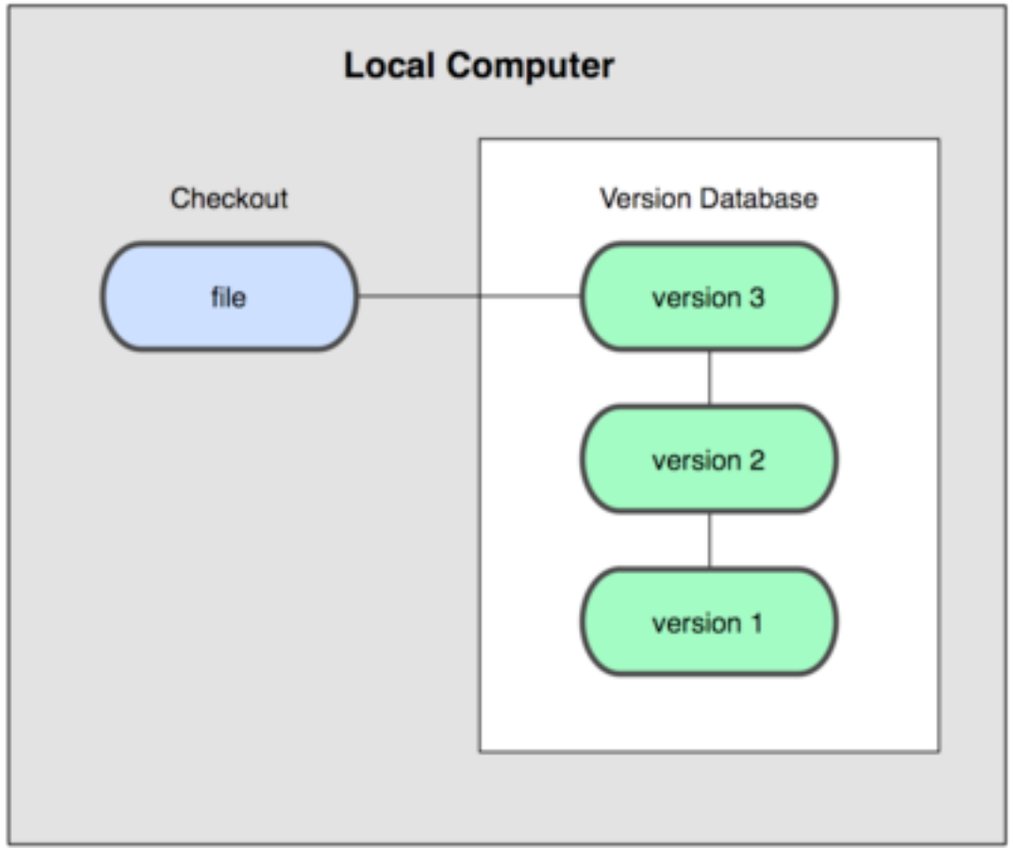
development and engineering

Version Control Systems

- Local Version Control Systems (SCCS, rcs)
- Central Version Control Systems (CVS, Subversion, Perforce)
- Distributed Version Control Systems (Git, Mercurial, Darcs, Bazaar)

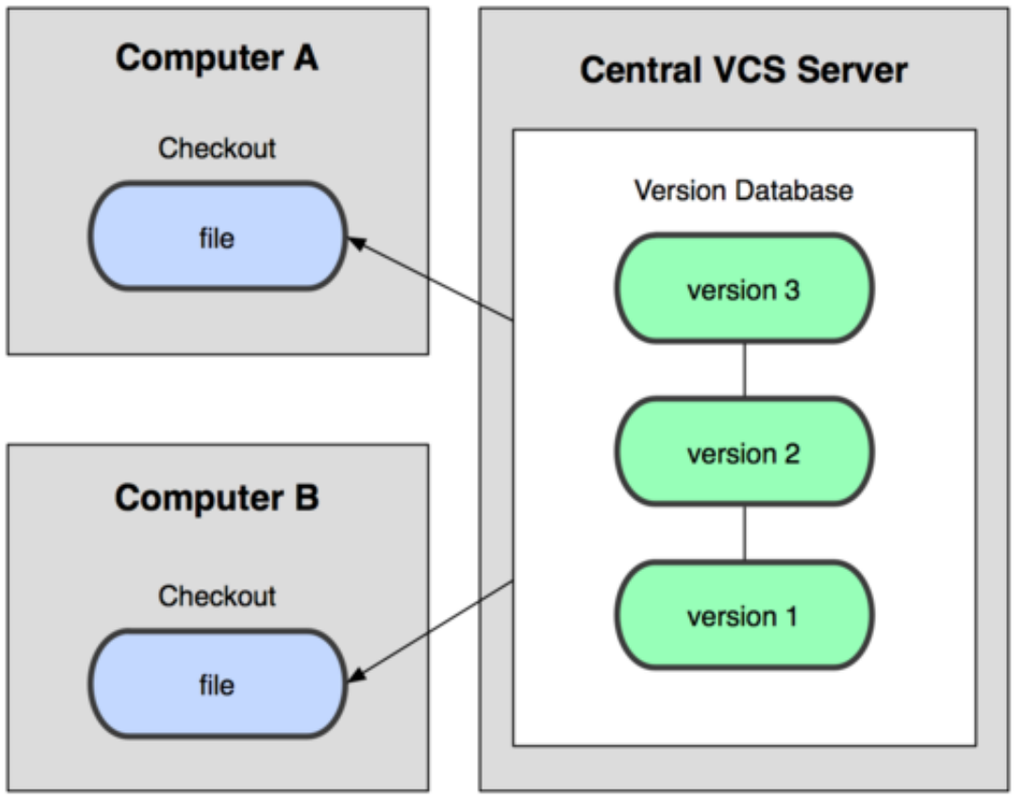
Local Version Control Systems

- Local Database
- Simple and Fast
- Single User
- Single Point of Failure
- No Collaboration



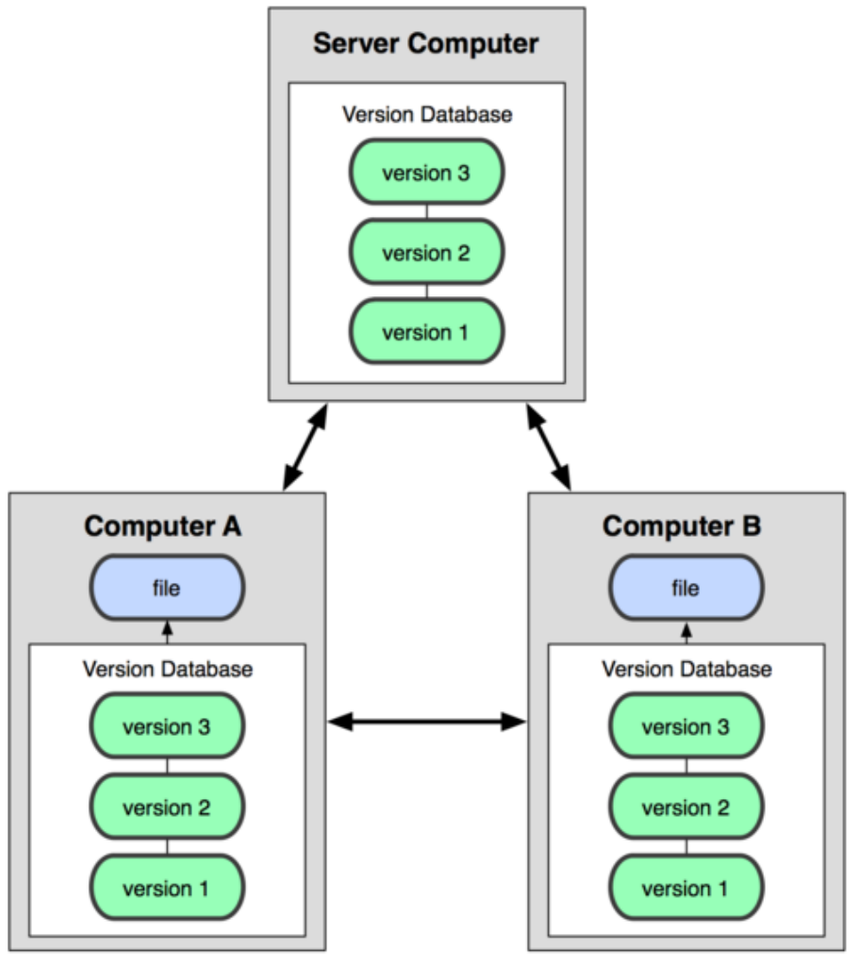
Centralized Version Control Systems

- Central Database
- Complex and Slow
- Multi User
- Single Point of Failure
- Limited Collaboration



Distributed Version Control Systems

- Distributed Database
- Complex and Fast
- Multi User
- No Single Point of Failure
- Unlimited Collaboration



A short history of Git

- 2002: Linux begin using proprietary DCVS BitKeeper
- 2005: BitKeeper revoked free-of-charge status
- April 2005: Initial release of Git by Linus
- December 2005:  1.0

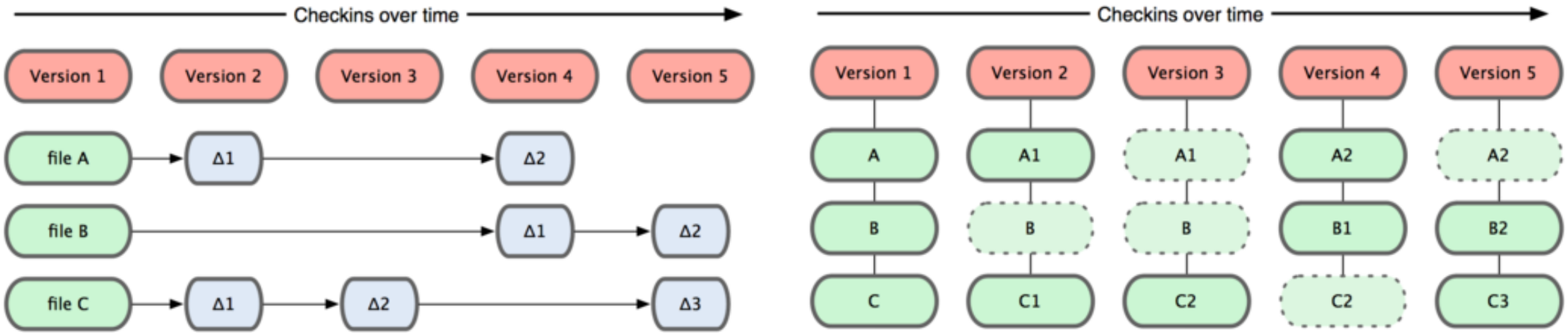
Design Goals of Git

- Speed
- Simple Design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently

Git Overview

- Snapshots, not Differences
- Nearly every Operation is Local
- Git Integrity
- Add-Only Workflows
- The Three States

Snapshots, not Differences



- Traditional approach to store file-based changes
- CVS, Subversion, Perforce, Bazaar

- Versions like snapshots of a mini-filesystem
- Distributed file system with powerful tools

Nearly every Operation is Local

- Most operations in Git only need local files and resources to operate
- Full history is locally available
- Network access only required to communicate changes with others
- No need to have access to central server all the time

Git Integrity

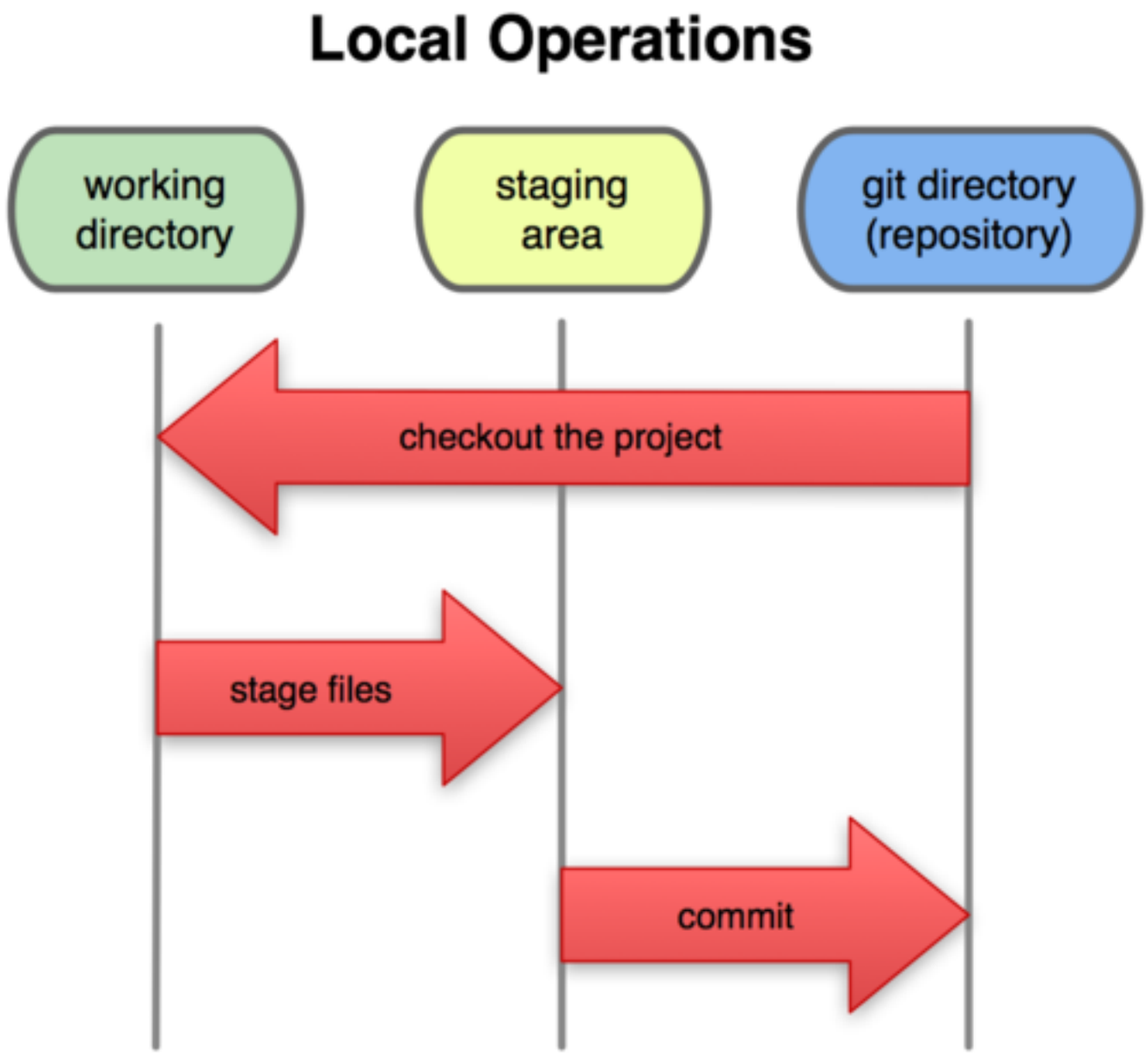
- Everything (really, everything!) in Git is check-summed using SHA-1
- No way to loose or corrupt data without Git being able to detect!
- On a related note: SHA1 check-summing is the key to Git's distributed operation

Add-Only Workflows

- Nearly all Git actions only add data
- Difficult to get the system to destroy data
- Once snapshot is committed, it's very difficult to lose
- Even lost data can be restored easily most of the time

The Three States

- Git adds a staging area (known as “the index”)
- Changes are added to the index first and once ready are committed to the (local) repository
- "If you deny the Index, you really deny git itself." (Linus Torvalds)



First-Time Git Setup

- Install Git

```
$ sudo apt-get install git-core
```

- Your Identity

```
$ git config --global user.name “Benedikt Meurer”
$ git config --global user.email “bm@os-cillation.de”
```

- Settings are stored in

`/etc/gitconfig`

`~/.gitconfig`

`.git/config` (in every repository)

Initializing a Repository in an Existing Directory

- Create a Git Repository

```
$ git init
```

- Git Repository is now in `.git`
- Import files

```
$ git add *.c
$ git add README
$ git commit -m “Initial import.”
```

- Files committed in `.git`

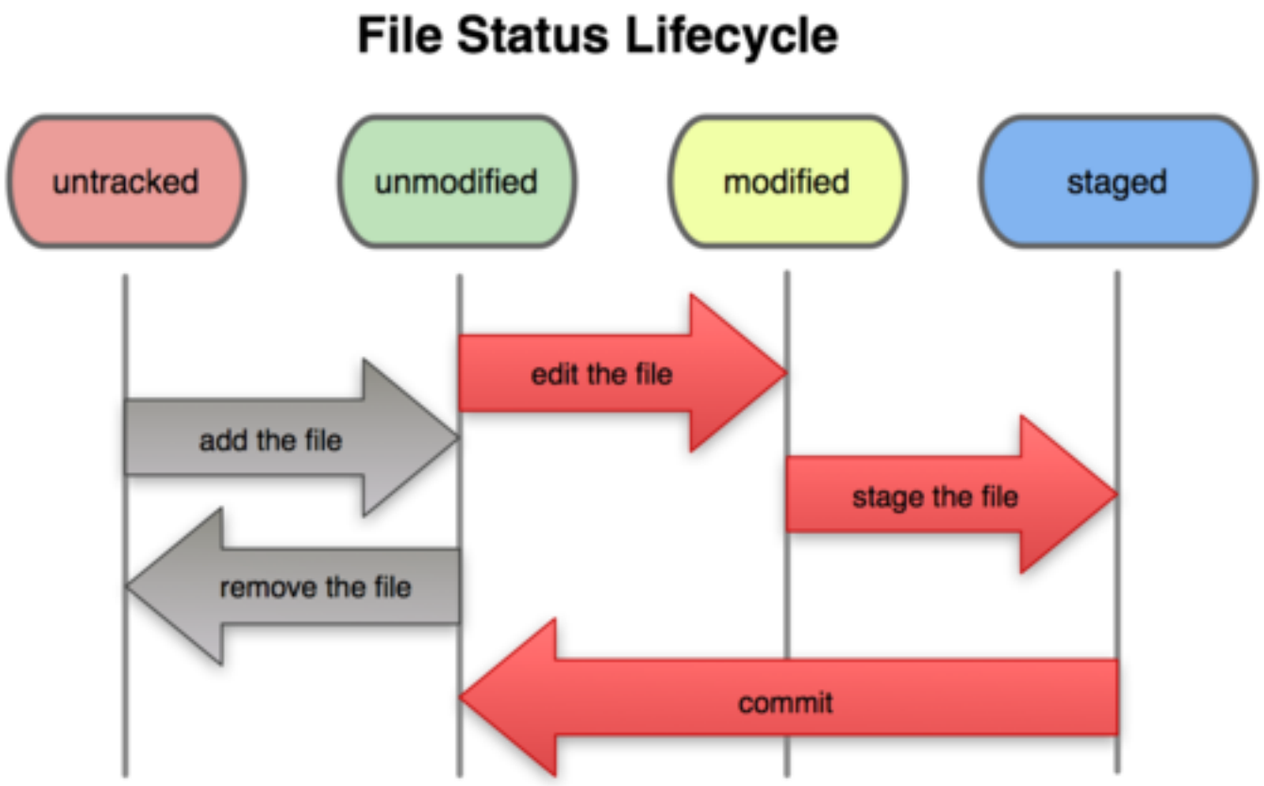
Clone an Existing (Remote) Repository

- Cloning takes a copy of a (remote) Git Repository

```
$ git clone git://core.os.de/os-cillation/testing.git
```

- Checkout in `testing`, cloned Repository in `testing/.git`
- This is **NOT** the same as `svn checkout`!

Recording Changes to the Repository



Checking Status of your Files

- Run initially after clone

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

- After some editing

```
$ touch test.c
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   test.c
nothing added to commit but untracked files present (use "git add"
to track)
```

Tracking New Files

- To begin tracking the test.c file

```
$ git add test.c
```

- Check status again

```
$ git status
# On branch master
# Changes to be committed
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   test.c
#
```

Committing Your Changes

- To commit all staged changes

```
$ git commit
```

- Different editor can be set using i.e.

```
$ git config --global core.editor vim
```

Working with Remote Repositories

- To push your changes upstream

```
$ git push -u origin master
```

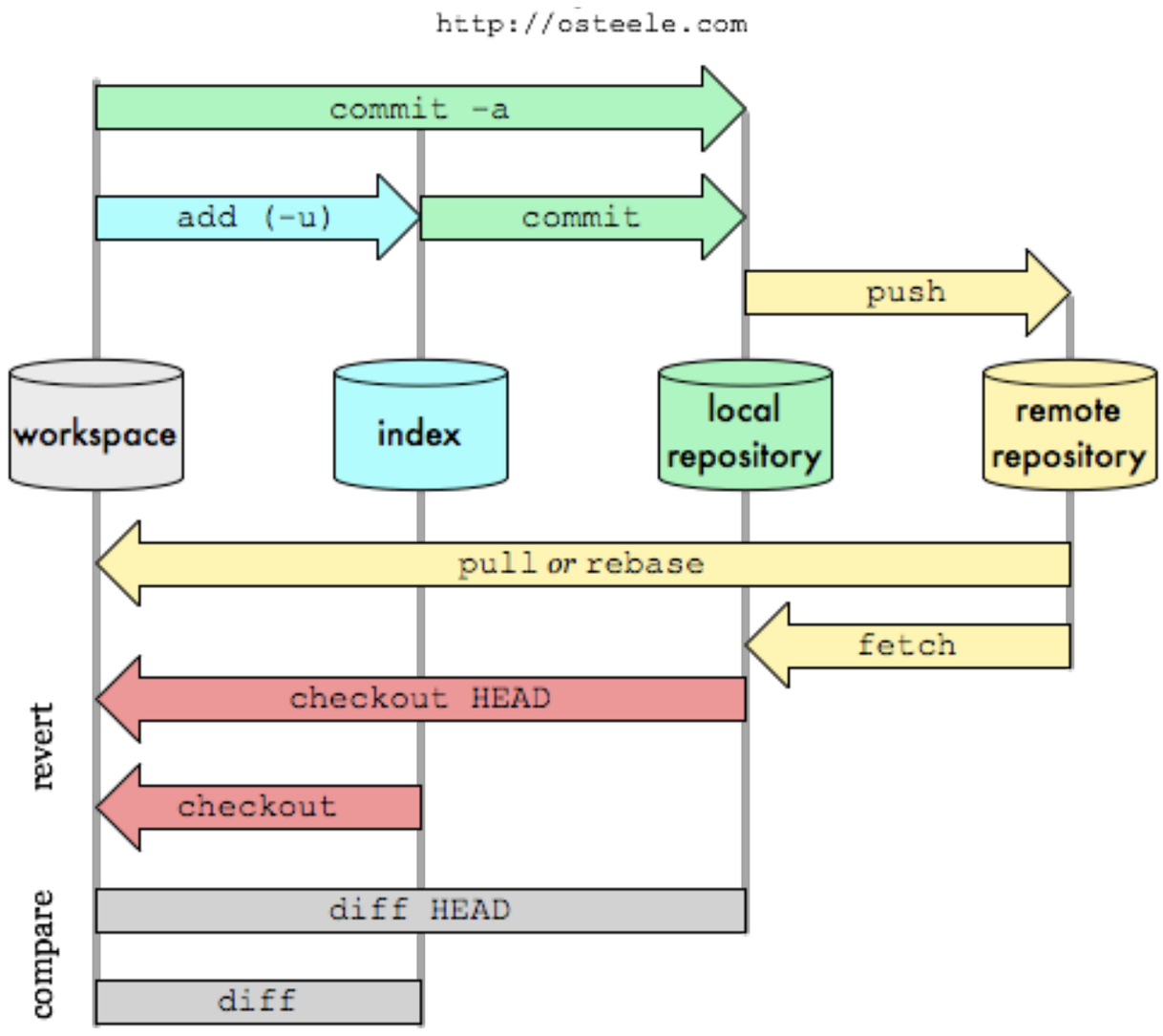
- To update all remotes

```
$ git remote update
```

- To pull changes from upstream

```
$ git pull origin master
```

Git Data Transport Commands



Git Setup

- Gitweb

  http://core.os.de/git/

- Clone URLs

  ssh://git@core.os.de/path/to/project.git

  git://core.os.de/path/to/project.git

- Playground

  os-cillation/testing.git

Git Tutorials / Documentation

- "Pro Git"

  http://git-scm.com/book

- "Git - SVN Crash Course"

  http://git-scm.com/course/svn.html

- "Visual Git Cheat Sheet"

  http://ndpsoftware.com/git-cheatsheet.html

Git Workflows

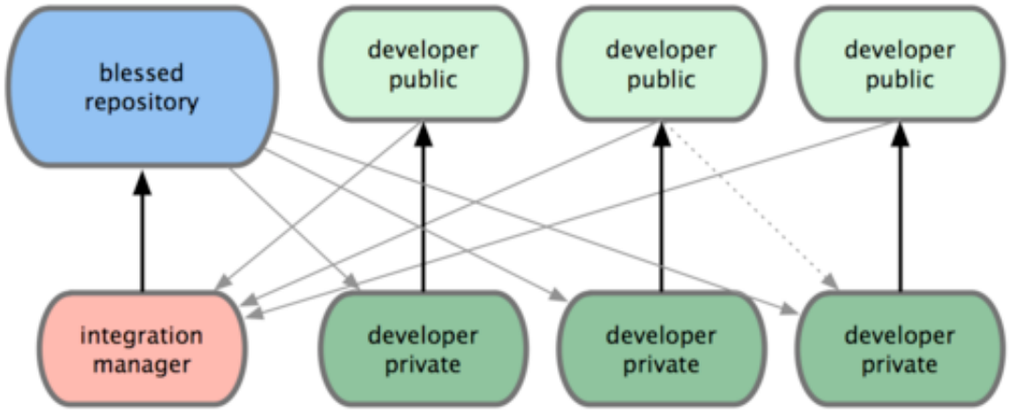
- Git is **NOT** a better Subversion!
- 1001 ways to shoot yourself in the foot!
- Define proper workflows to manage projects with Git

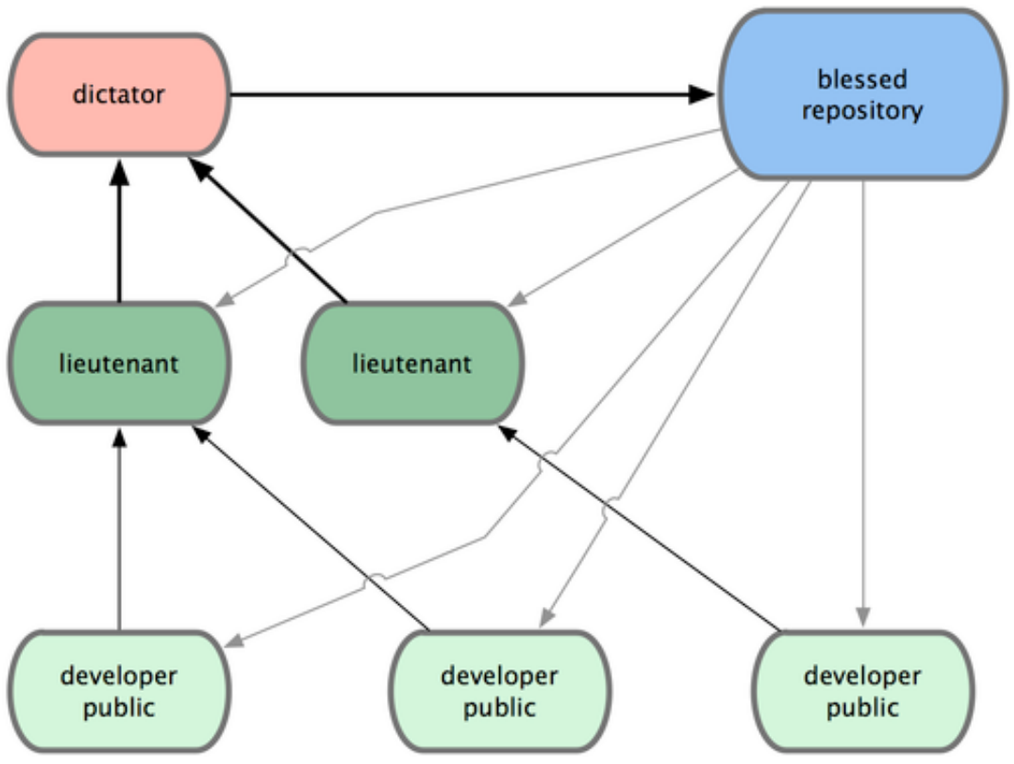
Git Workflows

- Workflows

  http://tinyurl.com/os-cillation-git-workflows

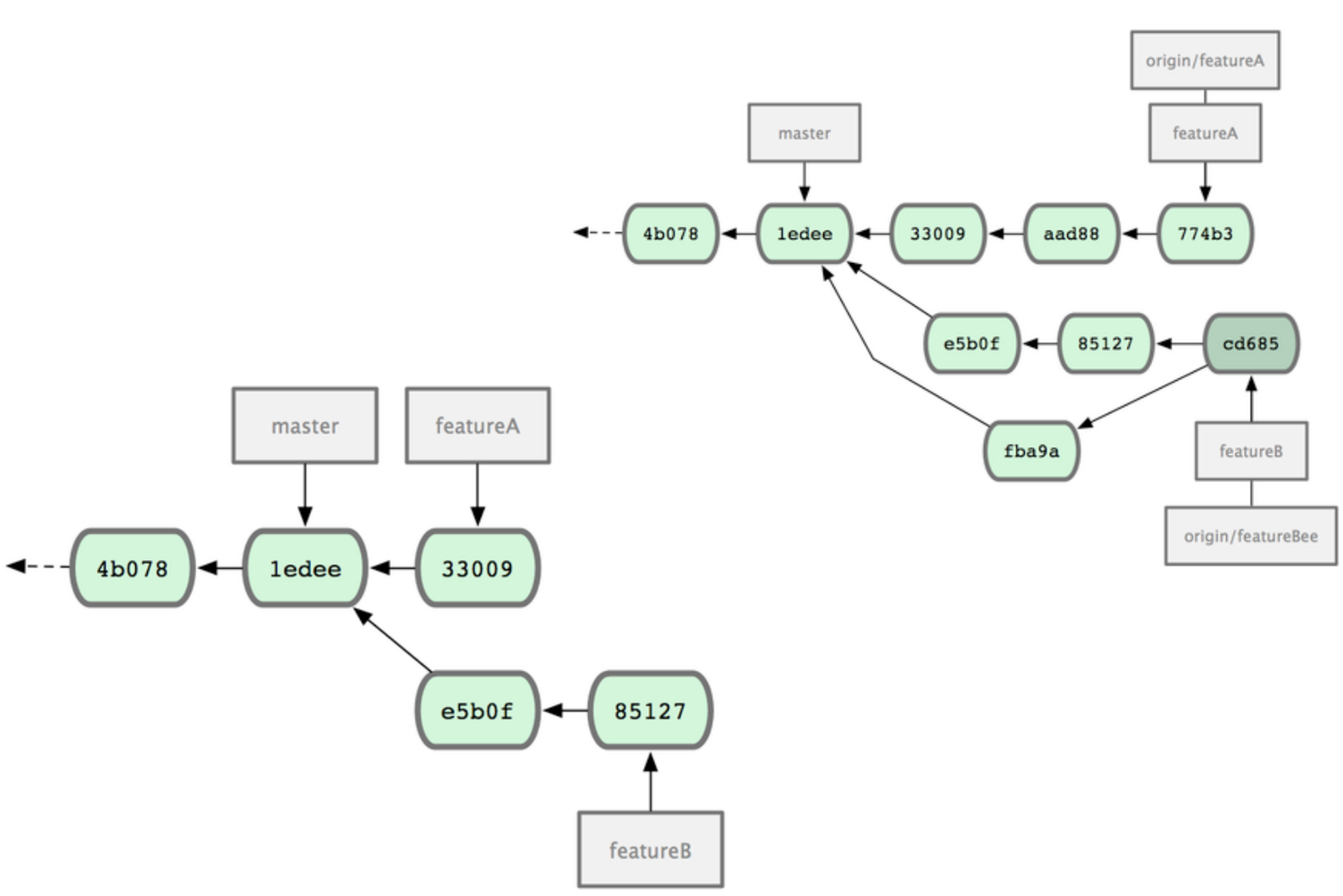
- Maintainers (Integrators) / Developers

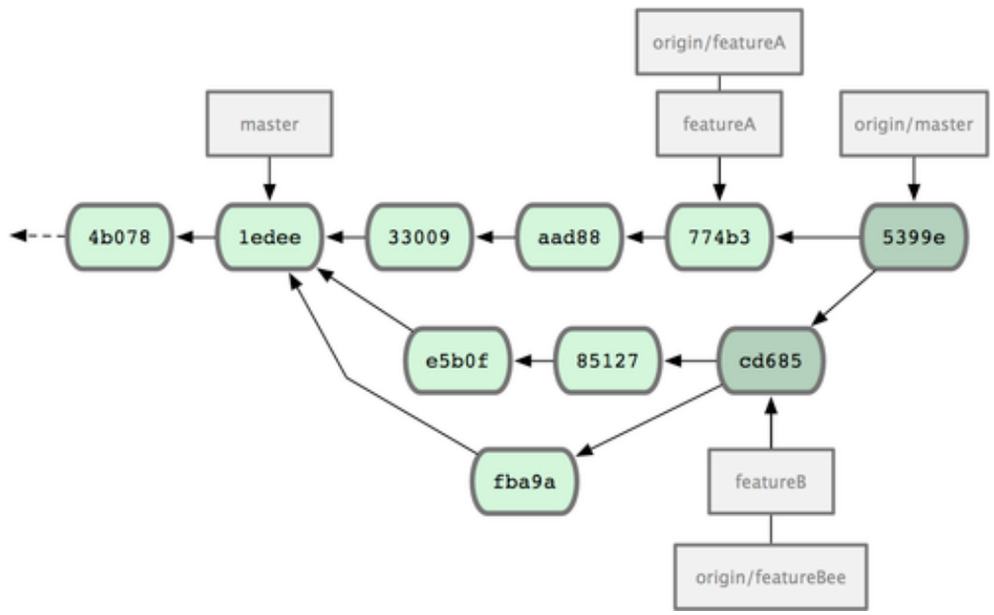




Git Workflows

- Development on Topic-Branches
- **DON'T** develop on the master Branch!





“Ok, got it, what now?”

- Setup Git
- Clone the testing Repository
- Familiarize with the basic Git commands (add, checkout, commit, branch, push, pull)
- Read and follow the workflow definitions
- Have fun and bitch at Subversion... ;-)