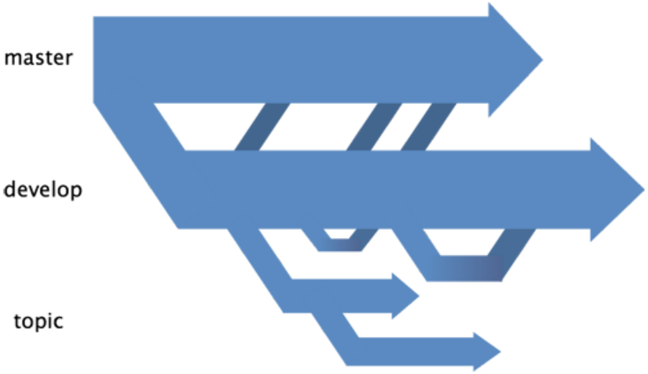
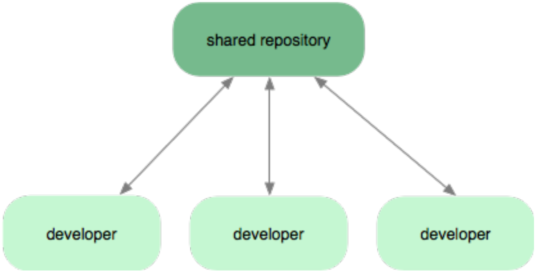
Hi mum!

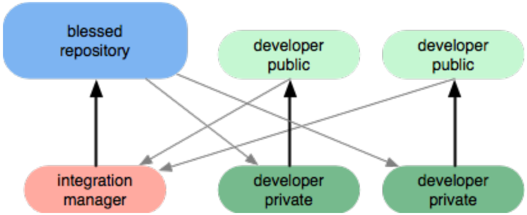
Killer feature: Cheap Local Branching



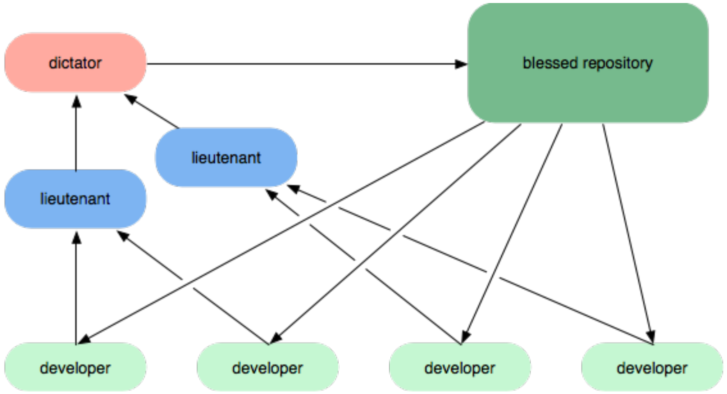
Supports any Workflow, Subversion-Style



Supports any Workflow, Integration Manager



Supports any Workflow, Dictator and Lieutenants



Pro: 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

1) "The following are a number of benchmarks that I performed on three copies of the Django source code repository"

2) add operation over 2000 files

Pro: fast 2

Info	Git	SVN	Magn.
Commit Files (A) - Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B) - Add, commit and push 1000 1k images	1.53	24.70	16x
Diff Current - Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent - Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags - Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50) - Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log (All) - Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log (File) - Log of the history of a single file (array.c - 483 revs)	0.60	82.84	138x
Update - Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame - Line annotation of a single file (array.c)	1.91	3.04	1x

Pros

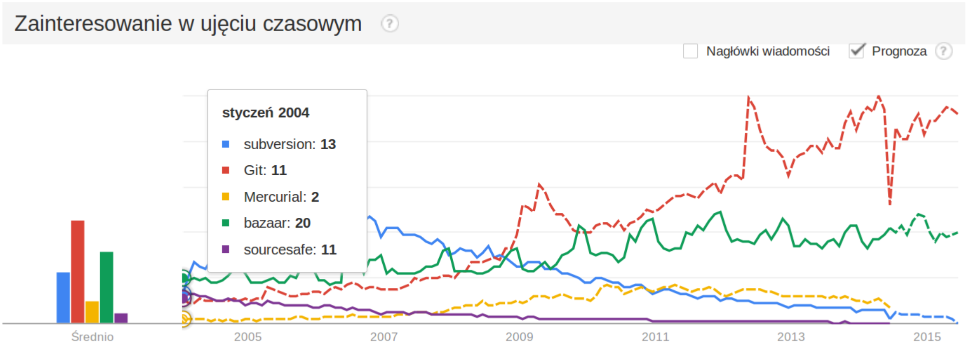
- Designed for distributed, non-linear, large-scale projects
- Makes 'take a small step' development very simple
- Improves clear communication about changes
- Encourages experimenting and contributing
- Implicit backup
- see: perfect commit in git

Pro: de facto standard 1

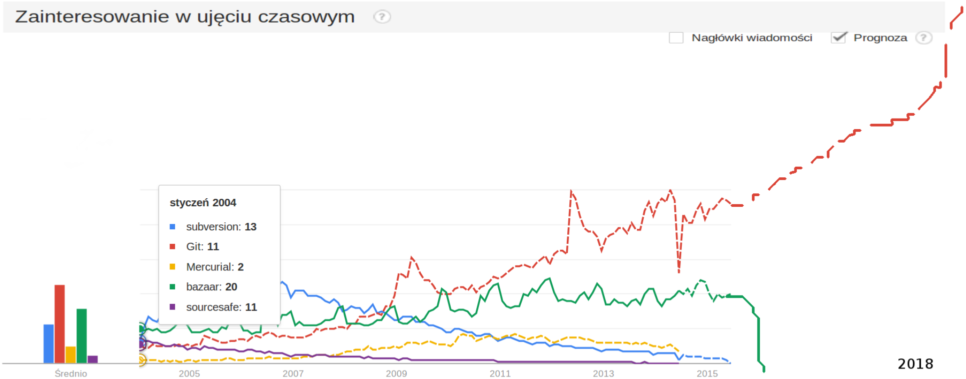
the most widely adopted version control system for software development

- Linux Kernel
- OpenVZ
- KVM
- Android
- Bacula
- dash
- Drupal
- FFmpeg
- GCC
- GNOME
- jQuery
- Mantis
- Openbox
- Perl5
- PulseAudio
- Puppet
- Qt
- Ruby on Rails
- VLC
- Wine
- x264
- YUI3
- Google
- facebook
- Microsoft
- twitter
- LinkedIn
- Netflix
- Apache Camel
- Eclipse
- PostgreSQL
- X
- KDE

Pro: de facto standard 2


Zainteresowanie w ujęciu czasowym

Pro: de facto standard 3



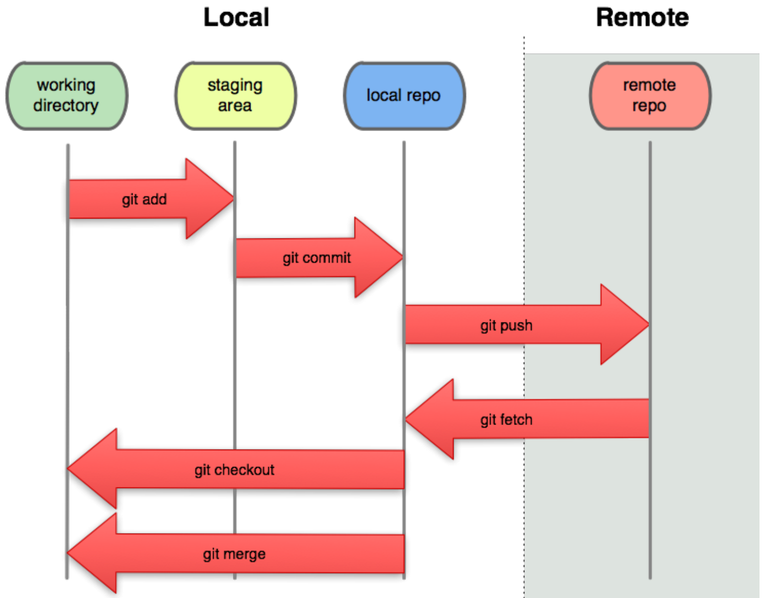
Cons

- Complex, 'controlling complexity is the essence of computer programming', Brian Kernighan
- Powerful, 'with great power comes great responsibility', Voltaire
- Cryptic command-line vocabulary
- Puts off integration

Contrast: SVN Pros

- Everybody knows it
- Simple
- Code integration happens quickly and often

Commit stages: overview



Commit stages: basic actions

- Commit represents changeset identified by hash
- Commit stages:
  - Untracked, not staged, working directory
  - Staging area, index
  - Repository, committed changes
- Remove from staging area: git reset
- Change/select commit to start working off: git checkout

Commit stages: checking status

- git status
- untracked vs. staging: git diff
- staging vs. repository: git diff --cached

DAG: Version graph

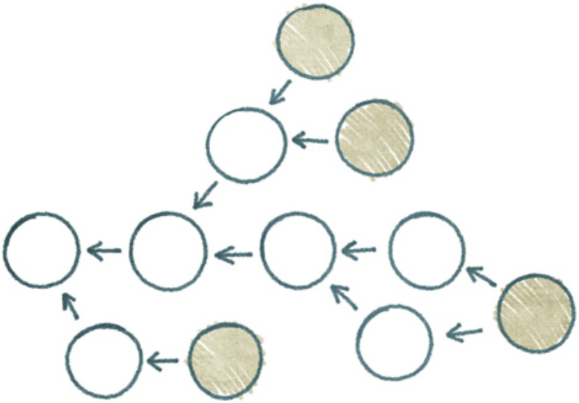


Figure 4.6. Not even close to being a Line

DAG: Version graph 2

- Node is a changeset
- Directed edge means "is based on"
- Branch is a pointer, master is a special branch
- HEAD is a pointer to branch pointer
- HEAD gives a view of the current version
- Detached HEAD means HEAD points directly at a node
- Commit with 2+ parents is a result of (non-ff) merge
- git log --graph: shows version graph

git commands

init

```
git init .
git init --bare .
```

clone

git clone URL

clone 2

- ssh://[user@]host.xz[:port]/path/to/repo.git/

easy to set up, authenticated	no anonymous access

- git://host.xz[:port]/path/to/repo.git/

fastest	lack of authentication

- http[s]://host.xz[:port]/path/to/repo.git/

easy to set up	inefficient, lot more network overhead

- ftp[s]://host.xz[:port]/path/to/repo.git/
- rsync://host.xz/path/to/repo.git/

- ssh://[user@]host.xz[:port]/ [user]/path/to/repo.git/
- git://host.xz[:port]/ [user]/path/to/repo.git/
- [user@]host.xz:/ [user]/path/to/repo.git/

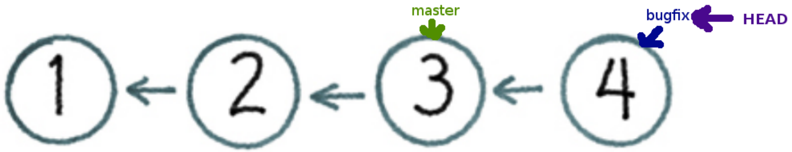
- /path/to/repo.git/
- file:///path/to/repo.git/

push

git push
git push repository
git push repository refspec

merge fast-forward 1

Having this:

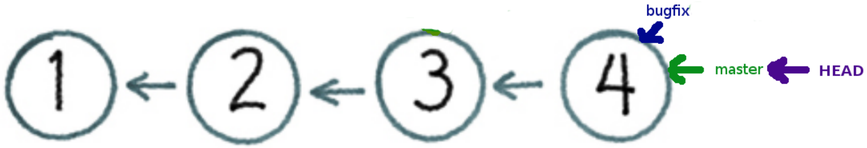


Let us merge:

```
git checkout master
git merge --ff bugfix
```

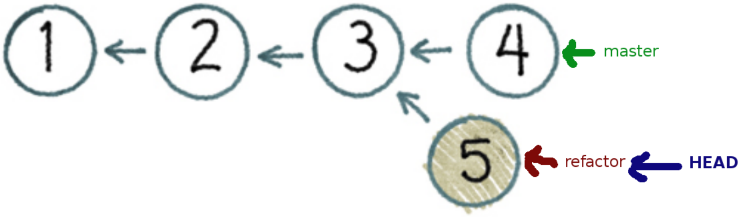
merge fast-forward 2

Result:



merge no-fast-forward 1

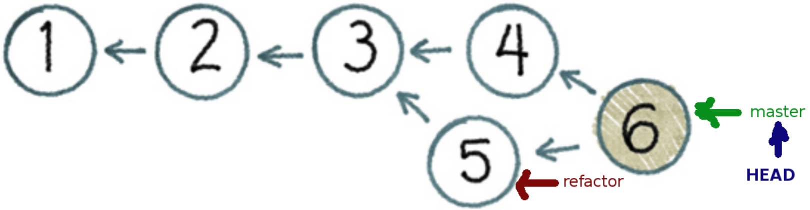
Having this:



Let us merge without fast-forward:
git checkout master
git merge --no-ff refactor

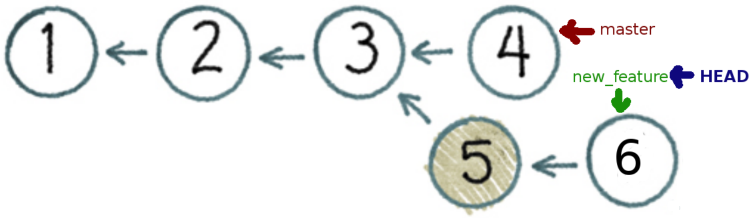
merge no-fast-forward 2

Result:



git rebase 1

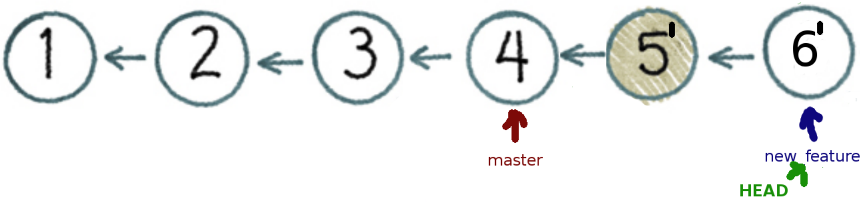
Having this:



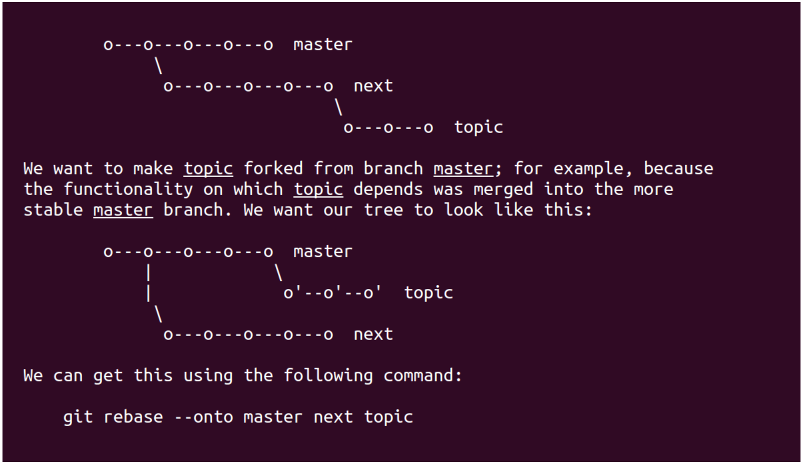
Let us rebase:
git checkout new_feature
git rebase master

rebase 2

Result:



rebase 3



```
        o---o---o---o---o  master
             \
              o---o---o---o---o  next
                               \
                                o---o---o  topic
```

We want to make topic forked from branch master; for example, because the functionality on which topic depends was merged into the more stable master branch. We want our tree to look like this:

```
        o---o---o---o---o  master
            |            \
            |             o'--o'--o'  topic
             \
              o---o---o---o---o  next
```

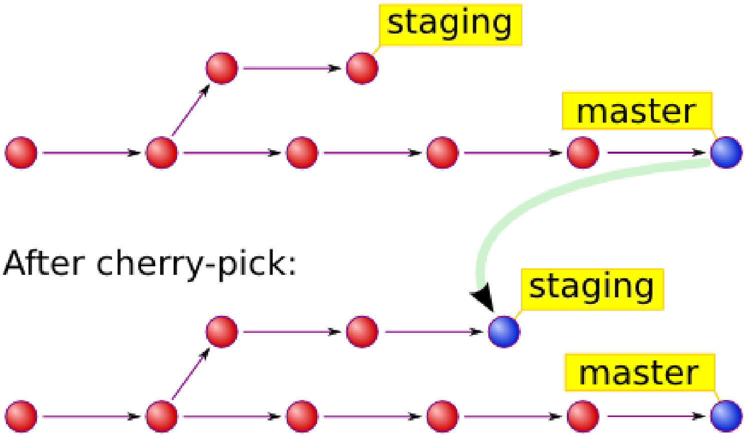
We can get this using the following command:

```
    git rebase --onto master next topic
```

rebase --interactive

- squash = combine commits
- reword = change commit message
- used to drop, reorder, split commits
- git rebase -i HEAD˜3 # change last three commits

cherry-pick



git checkout staging   git cherry-pick master

Workshop

Best practice

Best practice SM

Łukasz Rączka:

- binaries in git not recommended, repo slows down, hard to remove them
- git-svn: 4 attempts, not recommended, one person to merge repos (git and svn) through rebase
- separate repo per module, submodules: no good experience, semi-automatic solution

Best practice SM #2

Marcin Książek:

- use git central repo, add prefix for your repo to group all project repos
- biggest problem: people do not accept agreed rules, history re-write
- many smaller repos rather than one large
- SourceTree recommended (better version on Mac, no version on Linux)
- BeyondCompare - best but paid tool to merge (Lin/Win/Mac)

Best practice: others

- review changes before committing
- don't rewrite history
- small, logically concise commits:
  - new feature
  - formatting
  - other feature
  - no-op refactoring
  - bug fix
  - new API
- commit early and often

Best practice: others #2

- commit message: explain your changes:
  - first line: what?
  - third line: how? why? problem description, changes in architecture, solution limitations
- split work into repos
  - conceptually
  - according to permisions
  - large binaries
  - repo for history re-write
- don't panic, committed changes stay
- choose workflow

Quiz

- What are commit stages?
- What is commit?
- HEAD?
- Detached HEAD?
- Branch?
- DAG?
- Merge?
- Rebase?
- Rebase interactive?
- Cherry pick?

That's not all.

- Go through: LearnGitBranching
- Go through: GitHug
- Experiment on GitHub, GitLab, Stash
- Watch: Introduction to Git with Scott Chacon of GitHub
- Read free book: ProGit, Scott Chacon
- Find an expert