Git: Distributed Version Control Without Headaches

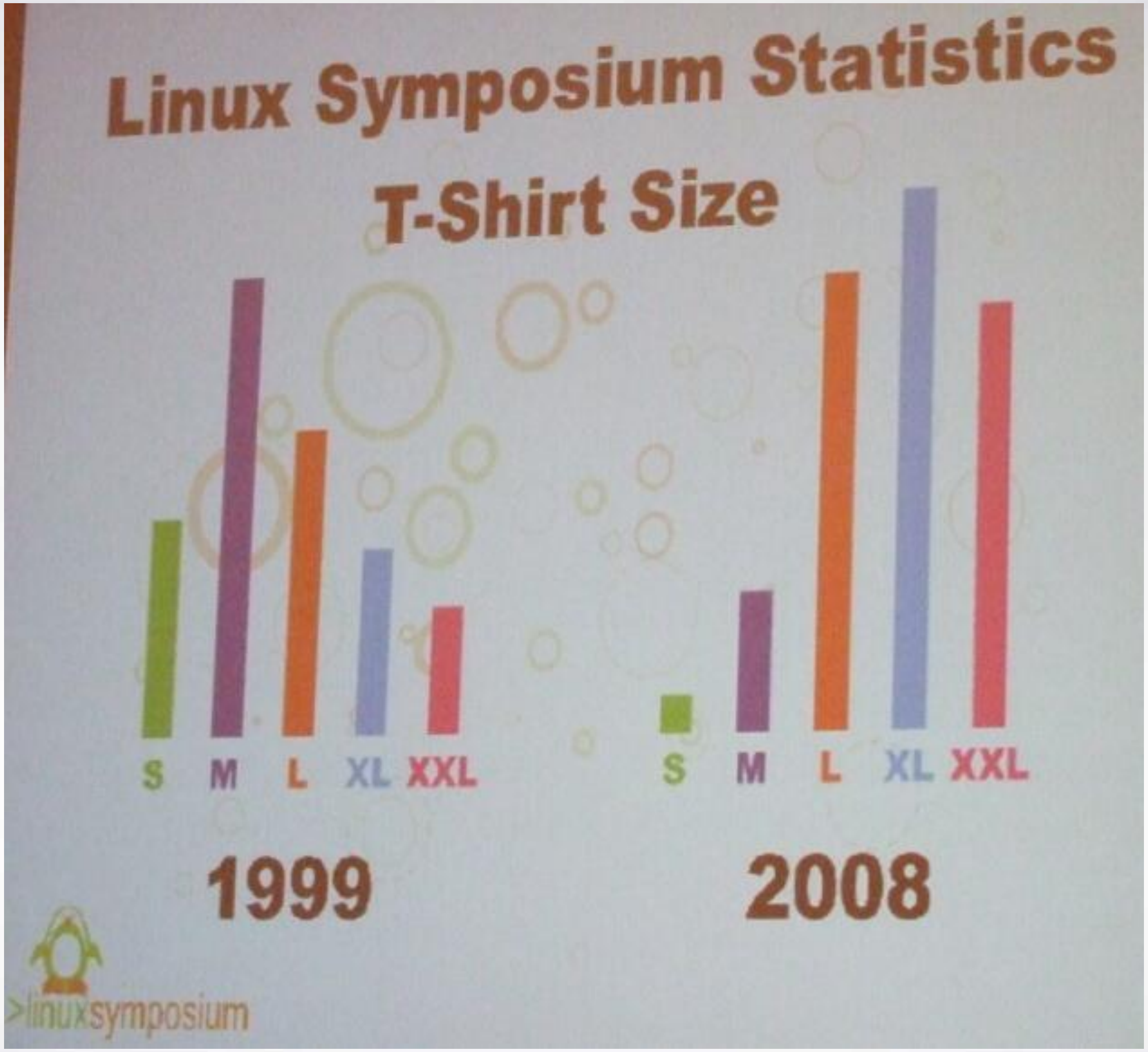
Bartosz Majsak, Thomas Hug

About Us

- Bartosz Majsak
  - Java Developer by day
  - Open source junkie by night (Arquillian core team member)
  - Conference speaker by passion (Devoxx, Jazoon ...)

- Thomas Hug
  - With Cambridge Technology Partners since 2002
  - Java Developer, TTL, Solution Architect
  - Apache Committer, OSS contributor and aficionado

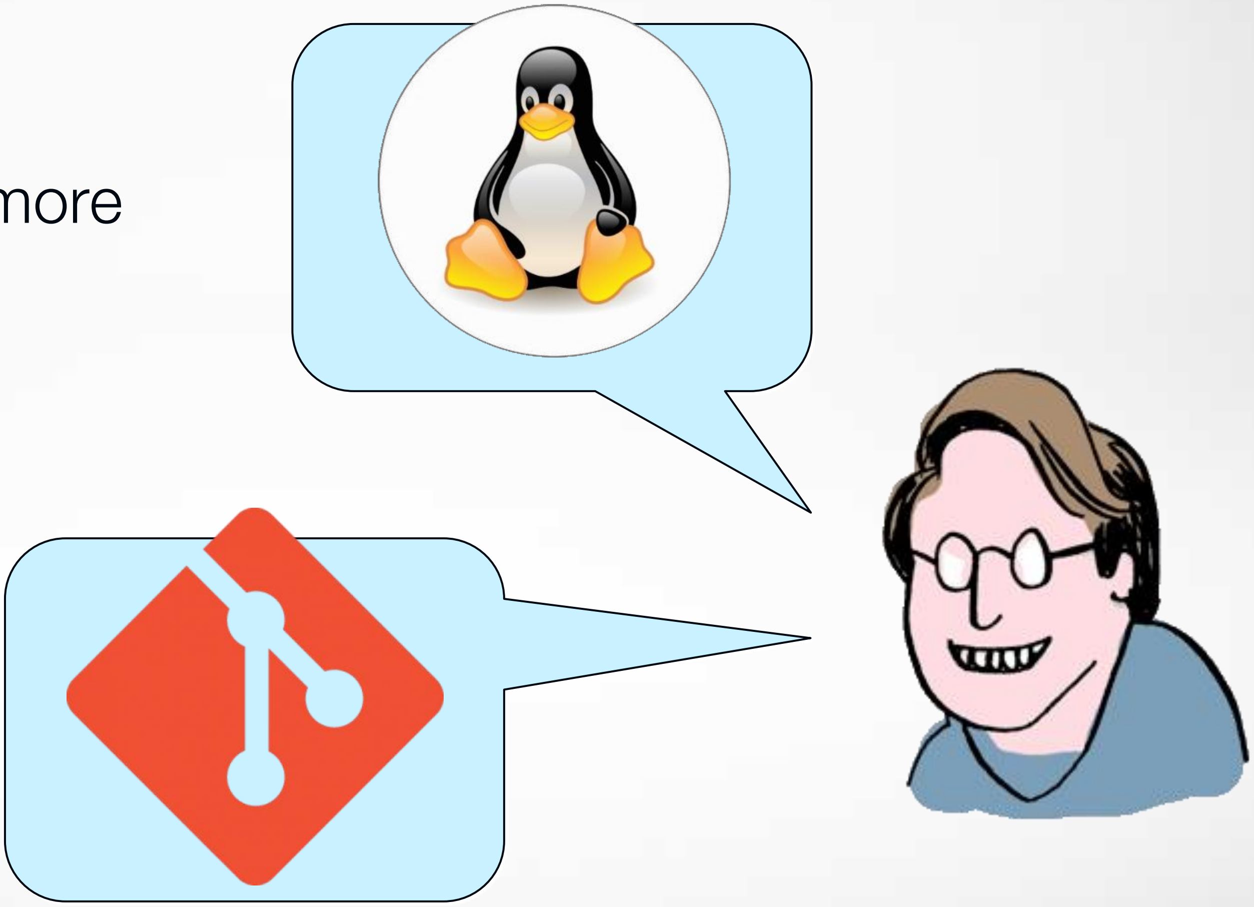
Why do we recommend Linux?



Git a British slang term meaning a contemptible person, a bastard.

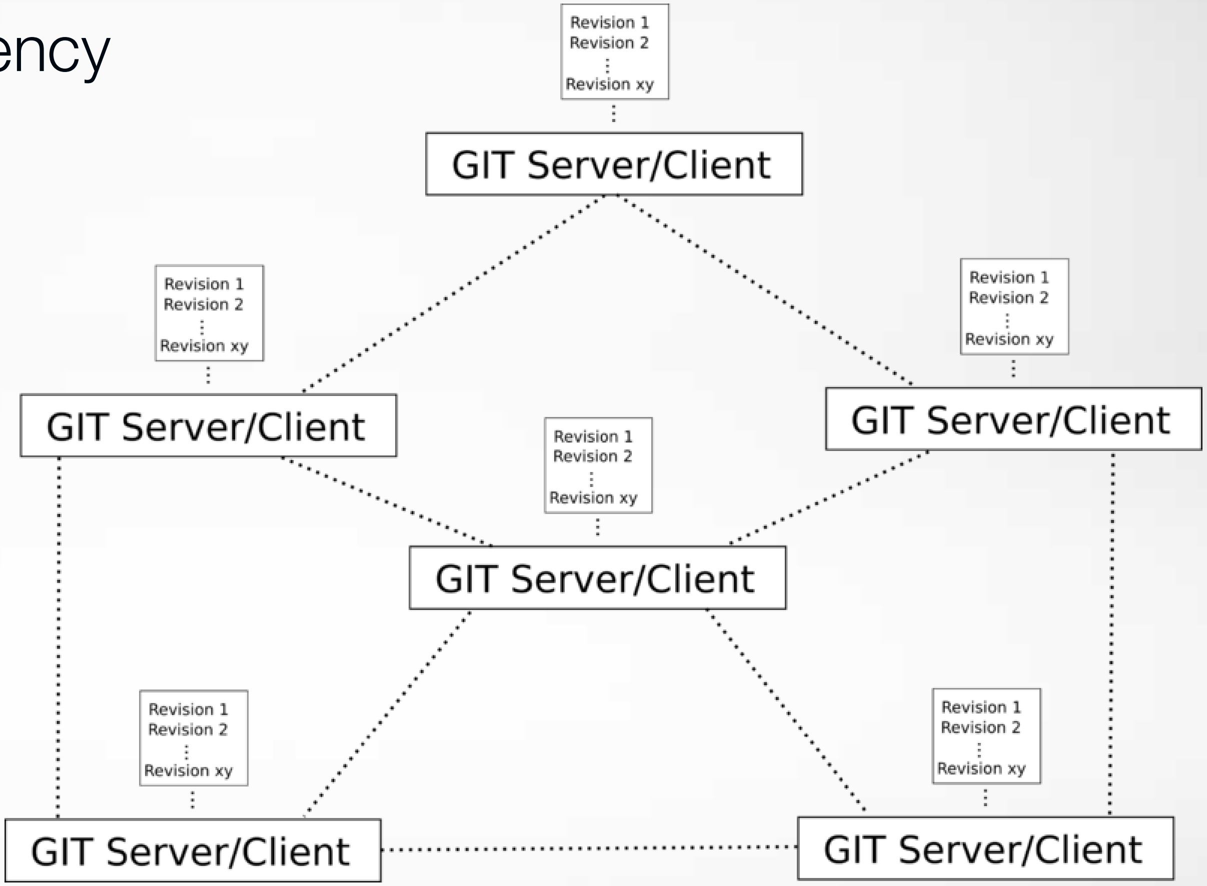
Git History

- Founded 2005 as a replacement of BitKeeper
- VCS of Linux Kernel
- …not just Linux anymore



Git Concepts

- No Central Server – Distributed VCS
- Performance and Efficiency
- Robustness



Disclaimer when we say **repository** we actually mean **local repository (no network connectivity)**

Installing and Configuring Git

Installation

- msysgit
- cygwin
- Atlassian SourceTree

- XCode
- Homebrew
- MacPorts

- Package Manager

Command line essentials

Playground

Objectives: getting familiar with essential commands and making your life easier

- touch
- cat / less
- mkdir
- ls
- tree
- cp / rm / mv
- nano
- history / ctrl+shift+r

User Identity

- Your contact details

```
$ git config --global user.name “Bruce Wayne”
$ git config --global user.email “batman@gotham.com”

$ less ~/.gitconfig
```

- SSH Key generation

```
$ ssh-keygen -t *dsa -C batman@gotham.com
```

**Using SHA-2 underneath. Approved by NSA*

Presets

- Color output

```
$ git config --global color.ui auto
```

- **Aliases**. Useful for stuff impossible to remember…

```
$ git config --global alias.showlog "log --color --
graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<%an>%Creset' --abbrev-commit"
```

Three levels of configuration:

--local (default, per repo) --global (per user) --system (machine)

Reference Material

- Git References
  - http://git-scm.com/ - official Git Home
  - http://git-scm.com/book - Pro Git (Apress) online version
  - http://git-scm.com/docs - Reference Documentation
  - https://www.atlassian.com/git/tutorial - Git Tutorial
  - http://gitready.com/ - Git Tutorial
- Workflows
  - https://www.atlassian.com/git/workflows - Tutorial on common Git workflows
  - http://yakiloo.com/getting-started-git-flow/ - About Git Flow (advanced topic)
- Getting Help
  - http://stackoverflow.com/ - All things programming
  - https://help.github.com/ - Git Recipies

First Repository

Creating a Repository

```
$ mkdir myrepo
$ cd myrepo
$ git init
$ git ls -la
```

git init

Do this in one swoop with

```
$ git init myrepo
```

Adding Files

```
$ touch index.html
$ git status
$ git add index.html
$ git status
```

git status

git add

`git add` works also with patterns:

```
$ git add '*.java'
$ git add .
$ git add folder/
```

You can even stage parts of a file

```
$ git add -p
```

Stage all changes (including deleted files) in the working directory with

```
$ git add -A .
```

Committing Files

```
$ git commit
$ git status
$ git log --oneline --decorate
```

git commit

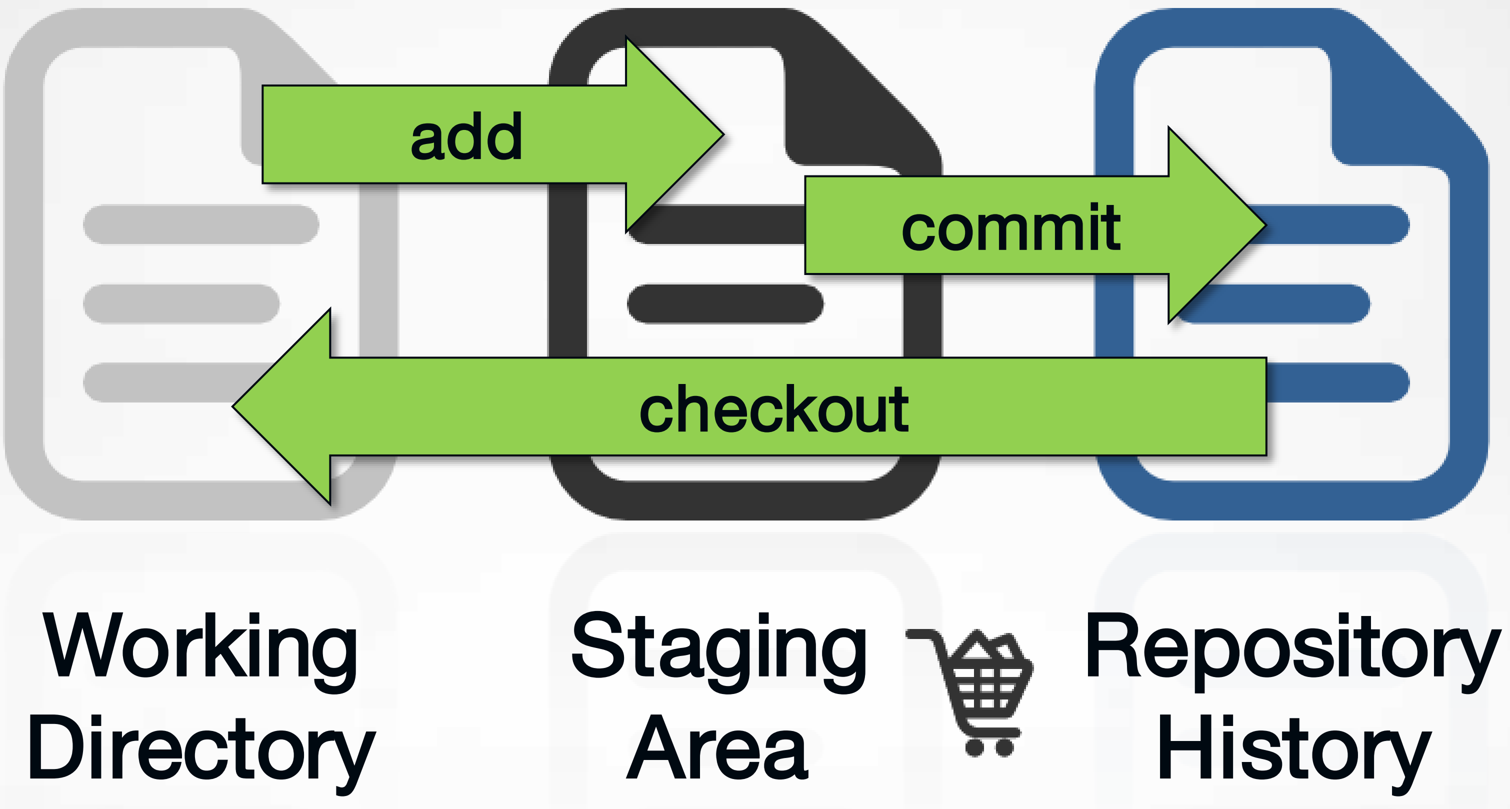
Commit directly with commit message:

```
$ git commit -m ‘Been there, done that’
$ git commit -am ‘Add also modified files directly’
```

Need a different commit editor?

```
export EDITOR=vim
```

Three Stage Thinking



Getting started

Objectives: learn what is needed to create git repository. Essential configuration.

- Initial configuration
- Create empty repository

```
$ git init
$ git status -sb
$ git config
$ git help <command>
```

Changes

Objectives: Learn what is staging area, what is "two-phase" commit in git and why is it useful to have this level of granularity.

- Create files
- Stage and commit them to the repository

```
$ git add <file> <folder/> <pattern>
$ git status -sb

$ git help <command>
```

Deleting and ignoring Files

```
$ touch test1.log test2.log
$ git add test1.log
$ git commit
$ vim .gitignore
$ git status
$ git rm test1.log
$ git commit
```

.gitignore

git rm

A shell script for easily accessing - gitignore boilerplates
https://github.com/simonwhitaker/gitignore-boilerplates

```
$ gibo Java Eclipse >> .gitignore
```

More stuff in!

Objectives: learn how change content of the repository by adding or removing the files.

- Add more files to the repository
- Modify and delete existing ones
- Exclude files and/or folders from the source control

```
$ git add
$ git commit -a -m”commit msg”
$ git status
```

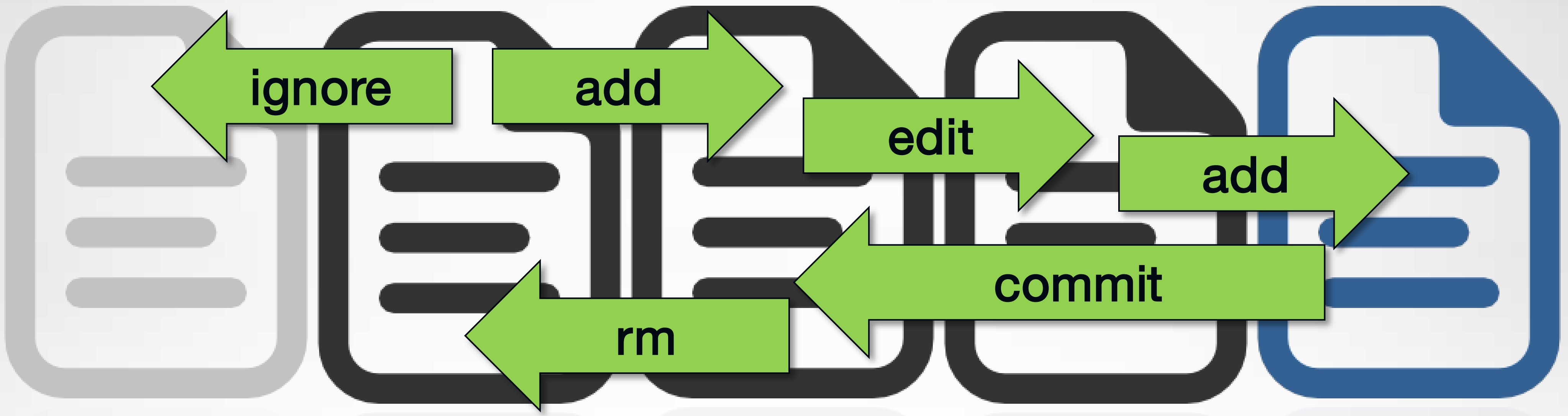
How does my repo look like?

`git log` gives you and overview of your repository structure.

```
$ git log
$ git log -p
$ git log --oneline --decorate

$ git log --graph --pretty=format:'%Cred%h%Creset
-%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<%an>%Creset' --abbrev-commit
```

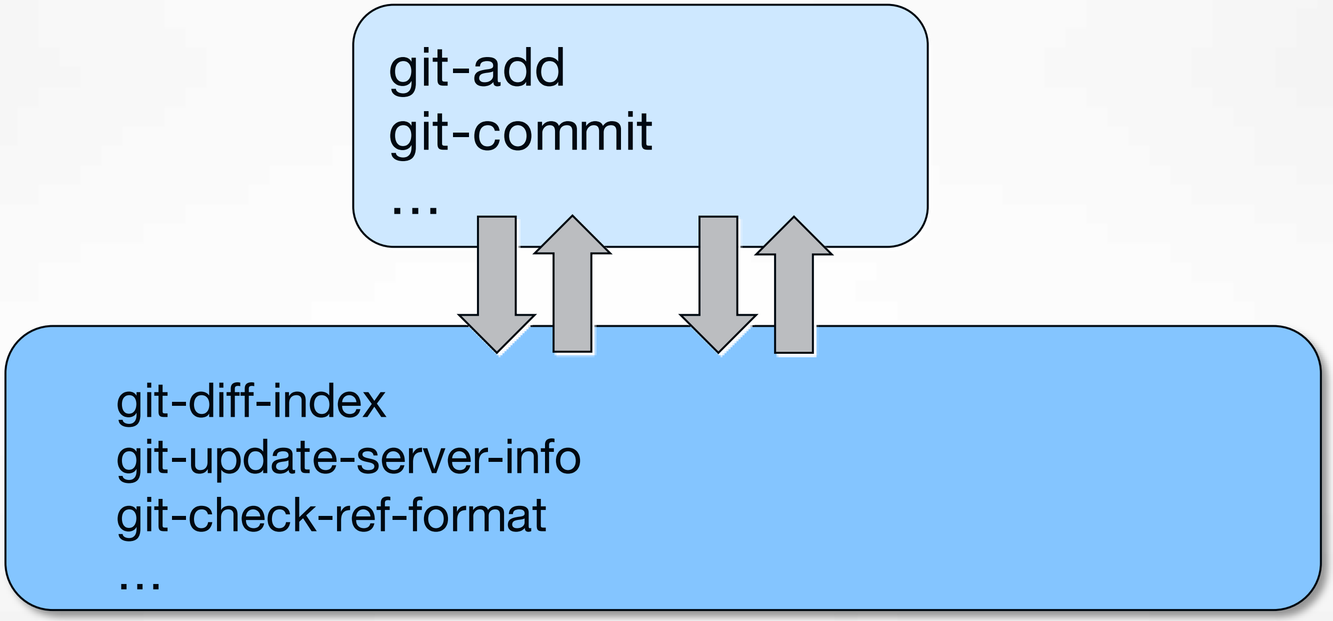
The Git File Workflow



Git Internals

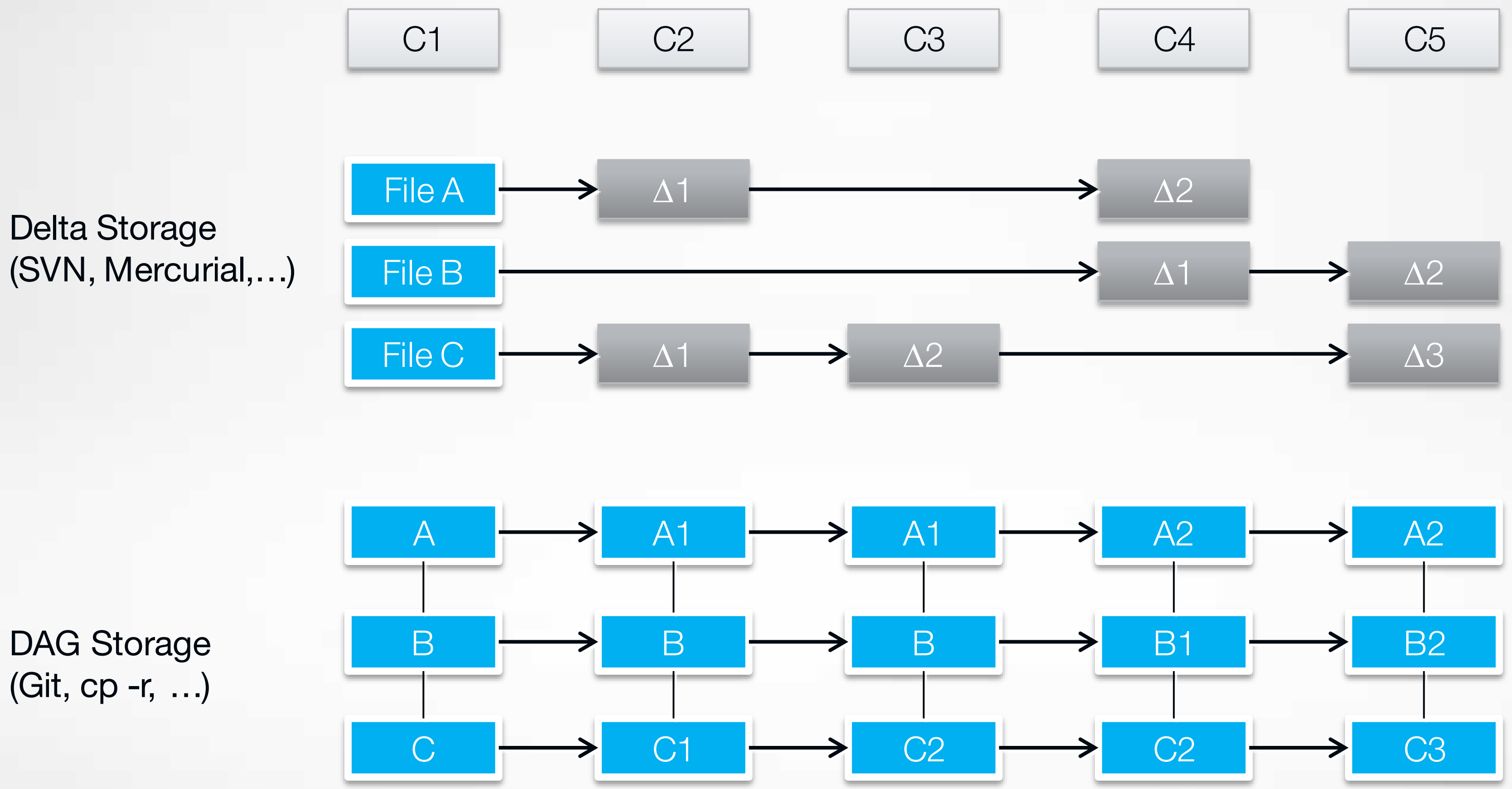
Git Architecture

- Plumbing and Porcelain
  - Composition of low-level commands into high-level ones
  - Unix design principles
- Local as much as possible



Git Storage

- Delta storage vs. Directed Acyclic Graph (DAG)

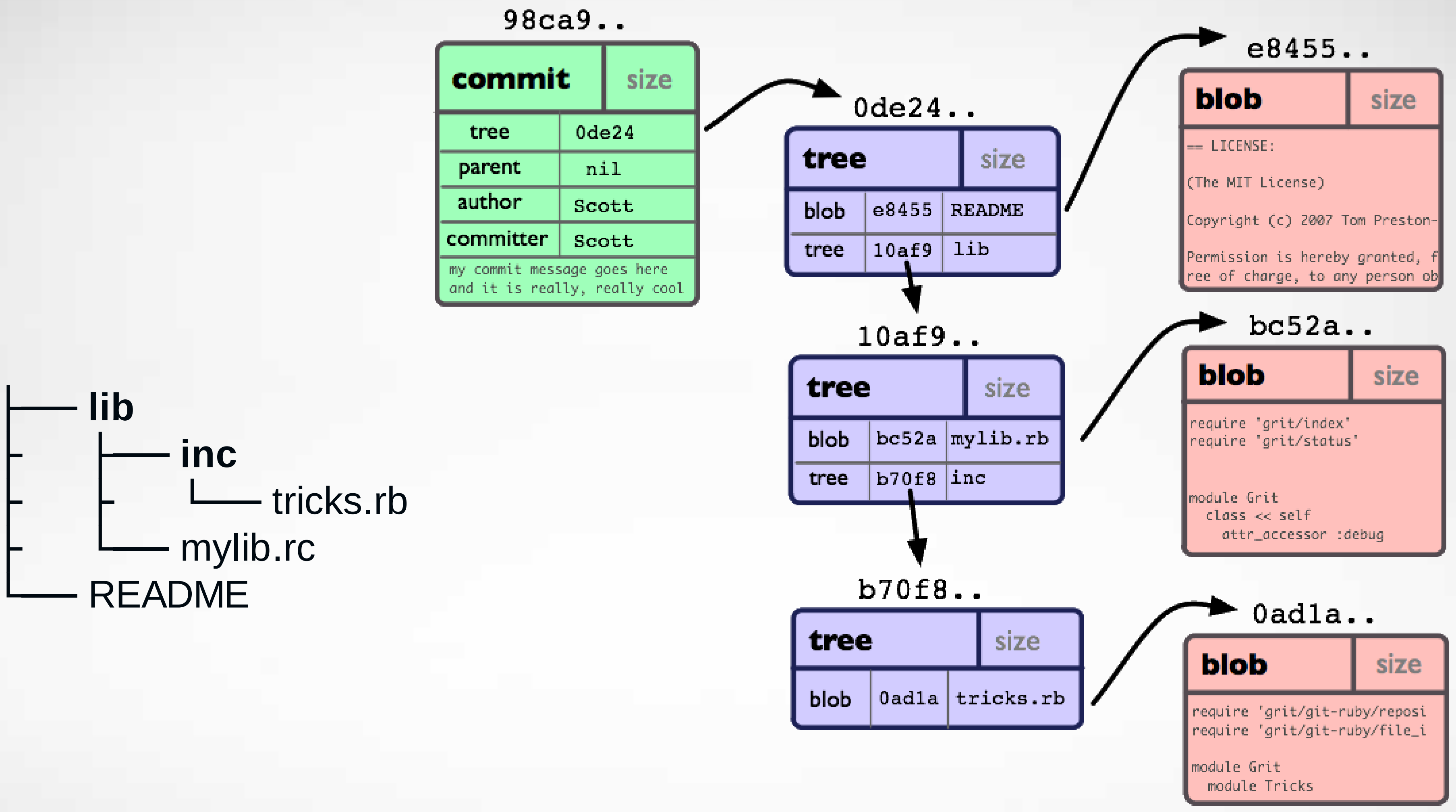


Git Storage – Object Model (1)

- Git tracks content, not files
- Content identified by 40 character SHA1 hash
  - Modified content easily identifiable
  - Immutable in the object database
- Objects: Blob, Tree, Commit, Tag
- References: HEAD, Tags, Remotes
  - Not immutable, pointers to commits

Empty directories are not considered as content.
Add an empty `.gitignore` if you need a folder tracked.

Git Storage – Object Model (2)



Git Storage – Local Repository

- The repository .git directory

```
$ cd .git
$ tree -L 1
├── branches        # Pointers to branches
├── config          # Repository local configuration
├── description     # Repository description
├── HEAD            # Pointer to HEAD in current branch
├── hooks           # Pre- and post action hooks
├── info            # Additional information about the repository
├── objects         # Object database
└── refs            # Pointers to branches
```

Branching and Merging

Branching

git branch

```
$ git branch mybranch
$ git branch
$ git checkout mybranch
```

Delete the branch with

```
$ git branch -d mybranch
$ git branch -D mybranch        # if unmerged
```

Create a branch and check it out in one swoop

```
$ git checkout -b mybranch
```

Time for some serious work

Objectives: Getting familiar with branching and tagging.

- Create new branch and modify repository
- Switch between branches
- Delete branch
- Create Tags

```
$ git branch
$ git checkout
$ git tag
```

Stashes

git stash

```
$ git status    # staged stuff
$ git stash
$ git status
…
$ git stash list
$ git stash apply [--index]
$ git stash drop stash@{0}
```

Apply and remove stash in one swoop

```
$ git stash pop
```

I'm not done yet

Objectives: Learn how to save local changes without the need to committing them (work in progress)

- Modify repository content and stage it
- Save as work in progress before switching the branch

```
$ git stash <TAB>
$ git help <command>
```

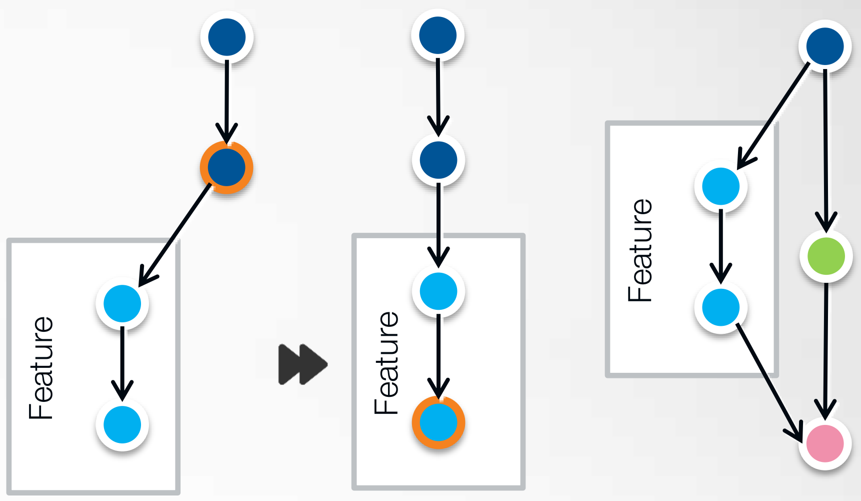
Merging

```
$ git checkout master
$ git branch mybranch
$ git showlog
$ git branch
```

Fast-forward is default

```
$ git merge --no-ff
```

git merge



Deactivating fast-forward merges per branch

```
$ git config branch.master.mergeoptions  "--no-ff"
```

Diffs

git diff

```
$ git diff mybranch master
```

Diff works also on the branch history

```
$ git diff                  # unstaged
$ git diff HEAD^^ HEAD      # from to
$ git diff hash1...hash2    # from to
```

Merging in action

Objectives: Learn merging changes from branches.

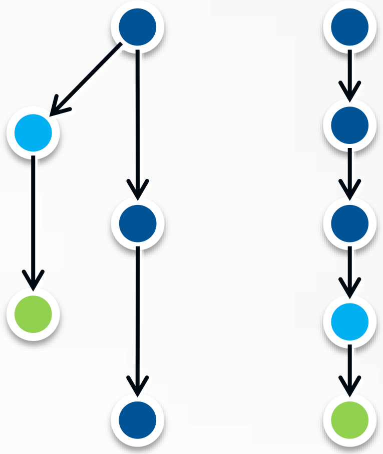
- Simple merge
  - Branch from master
  - Apply changes and commit
  - Merge back to master
- Resolving conflicts automatically
  - Branch from master
  - Apply changes on an arbitrary file and commit
  - Switch back to master and modify other file
  - Merge
- Conflicting changes

```
$ git checkout <BRANCH>
$ git merge
```

Rebasing

```
$ git checkout master
$ git rebase mybranch
```

git rebase



Rewriting history: Interactive rebase last four commits

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

All your base are belong to us

Objectives: Learn how rebase (interactive) works.

- Make changes on selected branch and rebase it with master
- Experiment with interactive rebase on selected branch
  - Reword commit messages
  - Combine commits into one

```
$ git rebase <BRANCH>
$ git rebase --i [commits range]
```

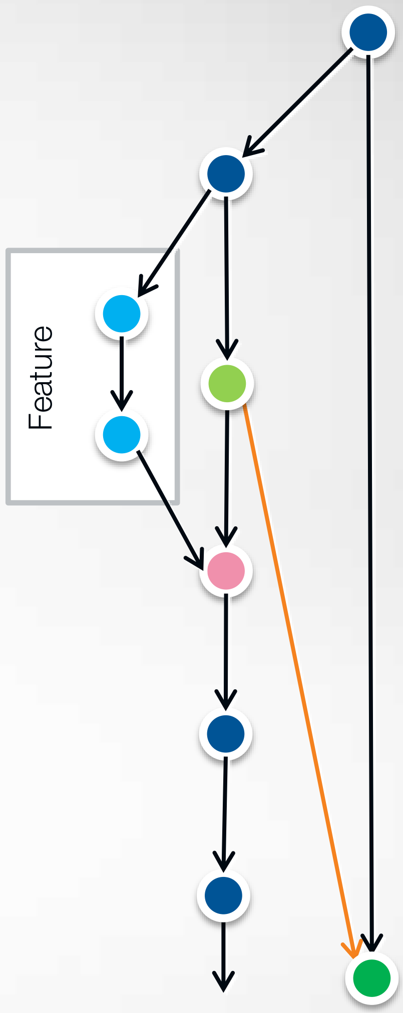
Cherry-pick

```
$ git cherry-pick [-x]
```

Cherry-pick „replays" arbitrary commits onto your current branch.

```
$ git cherry –v <other_branch>
```

Lets you check if given commit from other branch has been already applied on the current branch



Going Remote

Cloning Repositories

```
$ git clone [#remote]
```

git clone

Clone into a specific or existing (empty) folder

```
$ git clone [#remote] myclonedrepo
```

Git Protocols

- ssh / git: Securely connect to remote machines

```
git clone git@github.com:ctpconsulting/chopen-workshop-git.git
```

- HTTPS: Firewall friendly

```
git clone https://github.com/ctpconsulting/chopen-workshop-git.git
```

- File – simple. Can be used with e.g. a shared drive

```
git clone file:///home/thug/repo/chopen-workshop-git
```

Cloning directly without the file protocol will use hard links

```
$ git clone /home/thug/repo/chopen-workshop-git
```

Remotes

```
$ git init myremoterepo
$ cd myremoterepo
$ … # commit something
$ git remote add origin [#remote]
$ git remote -v
```

git remote

Git is distributed – you can have more than one remote!

```
$ git remote add https-origin https://myrepo.com/repo.git
```

Submitting Changes

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

git push

Forced push

```
$ git push --force
```

By default, Git always tries to push all matching branches.
Configuration to push only current to upstream:

```
$ git config push.default upstream
```

Retrieving Changes

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

Or short-hand

```
$ git pull
```

git fetch

git pull

Resolution strategy for merge conflicts

```
$ git pull -Xours
$ git pull -Xtheirs
```

It's all about collaboration

Objectives: Get familiar with remotes and GitHub

- Create github account
- Fork our repository
- Clone into your machine and add remote to ours (single source of truth)
- Let's play with branches and pull requests!

```
$ git clone
$ git push
$ git pull
```

Retrieving Changes

```
$ git pull --rebase
```

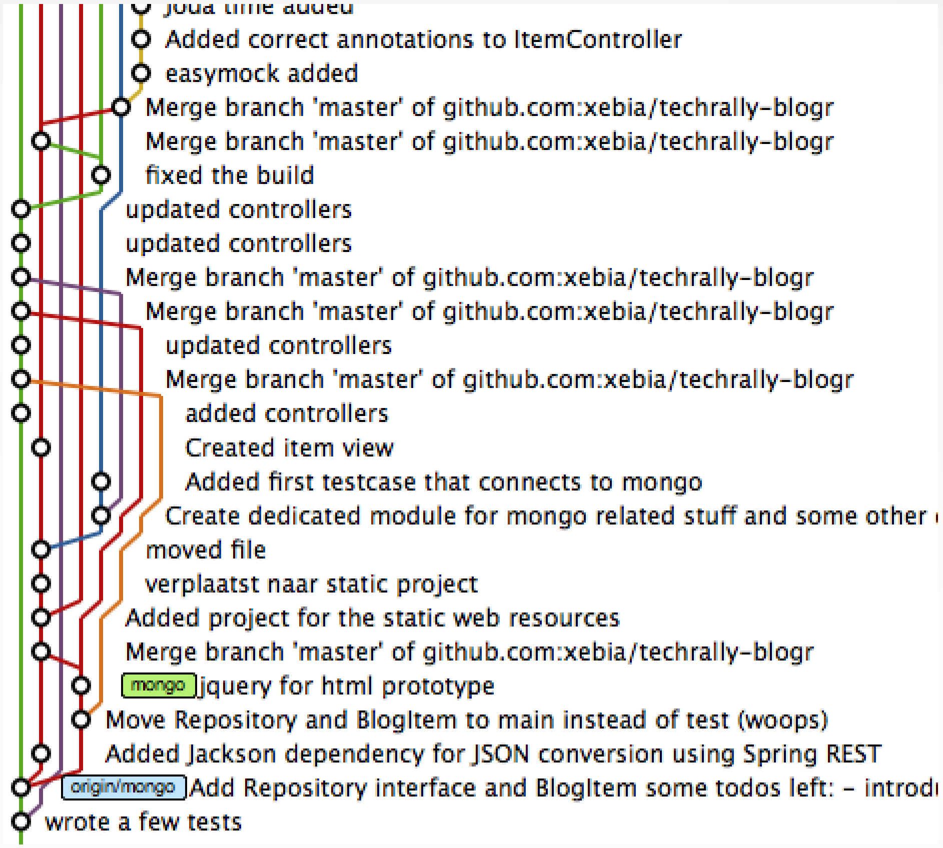
During a regular daily workflow where several team members sync a single branch often, the timeline gets polluted with unnecessary micro-merges on a regular `git pull`. Rebasing ensures that the commits are always re-applied so that the history stays linear.

Make `git pull` on master always use rebase

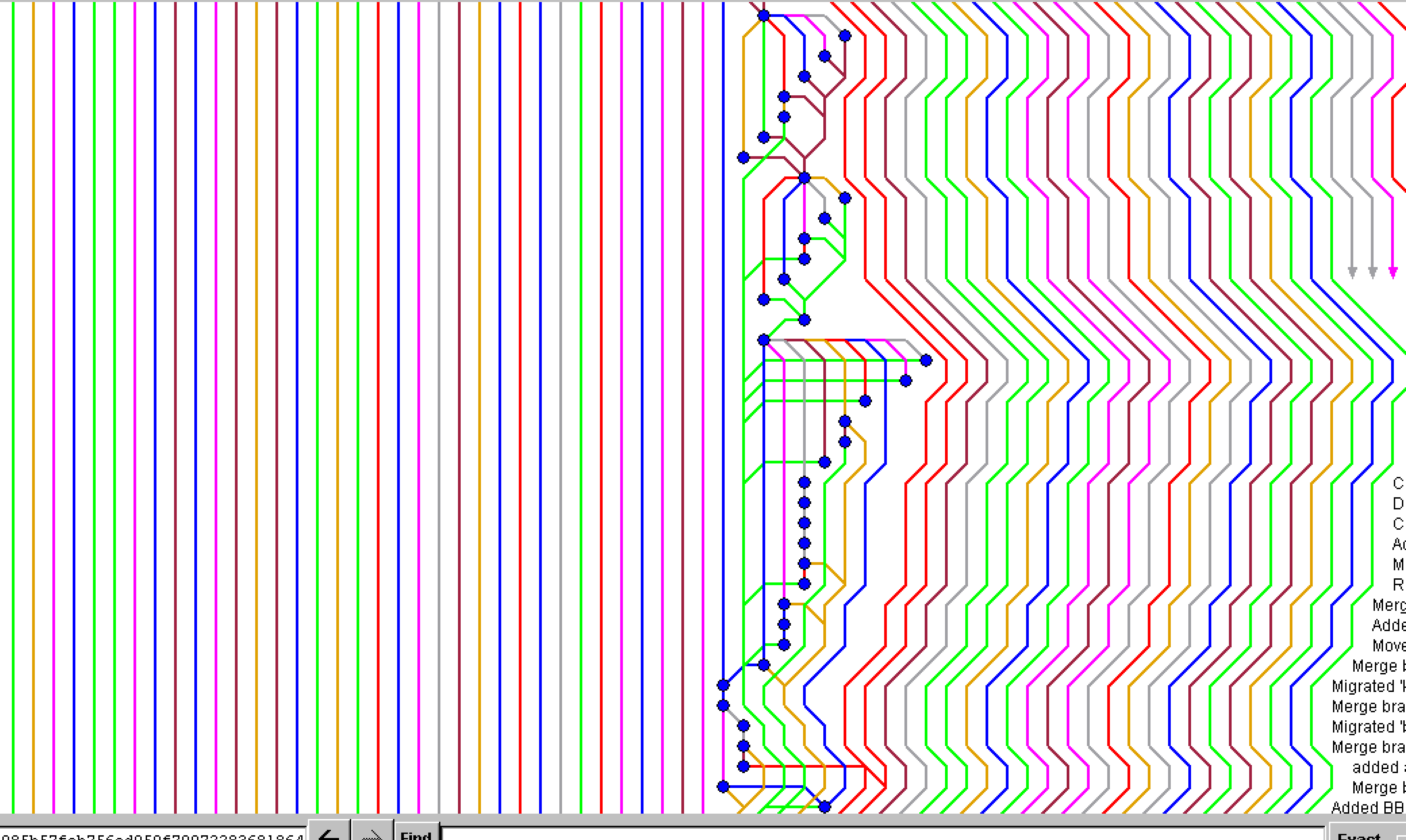
```
$ git config branch.master.rebase true
```

Or make it a default for every tracking branch strategy

```
$ git config --global branch.autosetuprebase always
```



joda time added
Added correct annotations to ItemController
easymock added
Merge branch 'master' of github.com:xebia/techrally-blogr
Merge branch 'master' of github.com:xebia/techrally-blogr
fixed the build
updated controllers
updated controllers
Merge branch 'master' of github.com:xebia/techrally-blogr
Merge branch 'master' of github.com:xebia/techrally-blogr
updated controllers
Merge branch 'master' of github.com:xebia/techrally-blogr
added controllers
Created item view
Added first testcase that connects to mongo
Create dedicated module for mongo related stuff and some other
moved file
verplaatst naar static project
Added project for the static web resources
Merge branch 'master' of github.com:xebia/techrally-blogr
mongo jquery for html prototype
Move Repository and BlogItem to main instead of test (woops)
Added Jackson dependency for JSON conversion using Spring REST
origin/mongo Add Repository interface and BlogItem some todos left: - introd
wrote a few tests


C
D
C
Ad
M
R
Merg
Adde
Move
Merge b
Migrated '
Merge bra
Migrated '
Merge bra
added
Merge b
Added BB
Find
Exact

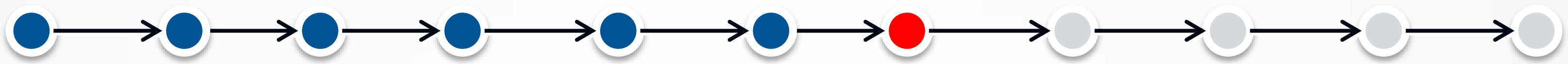
Who broke the build?!

```
$ git blame FILE

$ git bisect start
$ git bisect bad
$ git bisect good <HASH>
```

git blame

git bisect



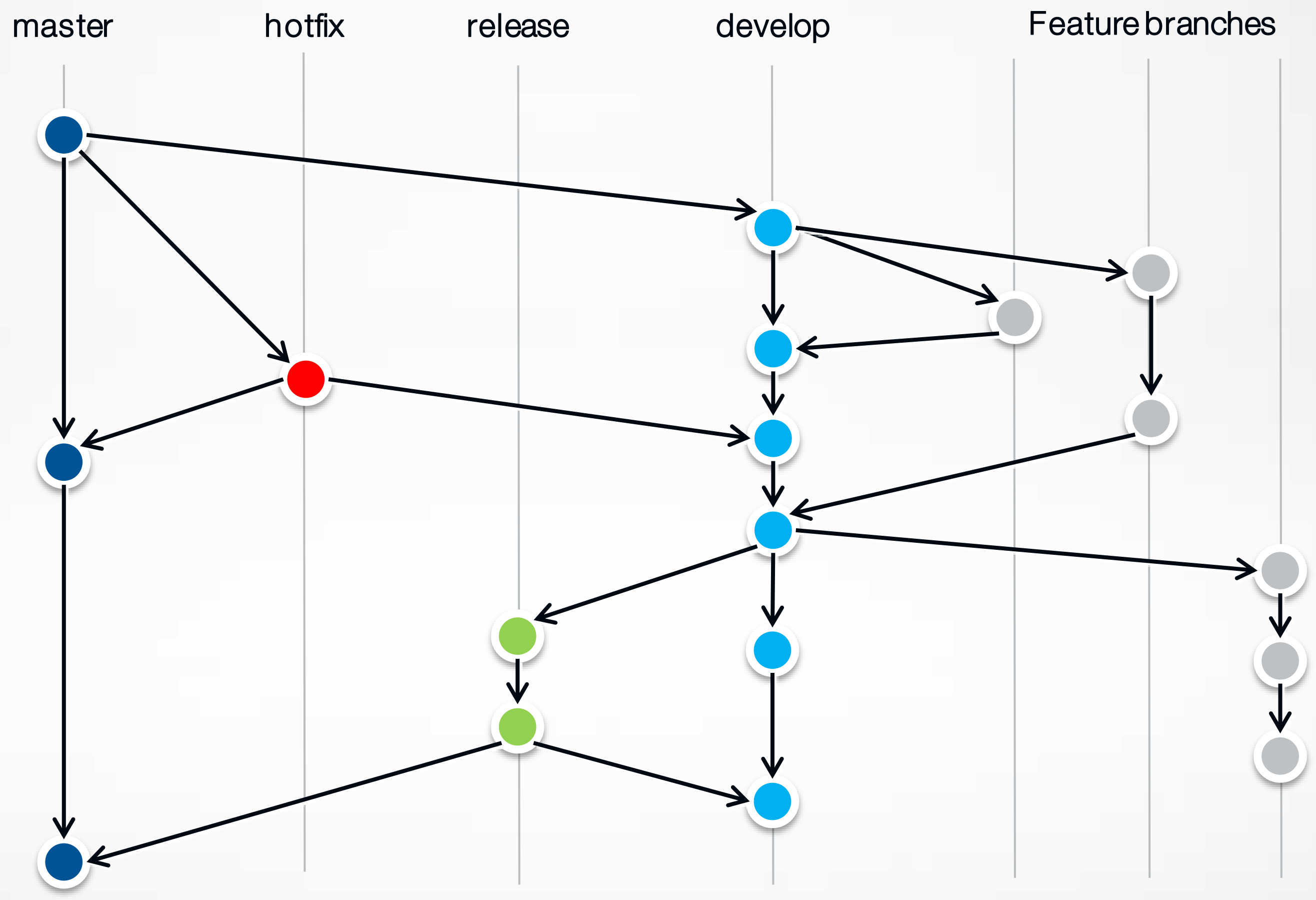
Cambridge Technology Partners

Workflows

Git Flow

```
$ git flow init
```

git flow



Cambridge Technology Partners

Git in the Enterprise

Recovering from Mistakes

Fixing Commits and Staging Area

git reset

For not yet pushed commits:

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

Unstage a file:

```
$ git reset HEAD file.txt
```

Discard local changes:

```
$ git checkout -- file.txt
```

Fully revert to a previous commit:

```
$ git reset --hard HEAD
```

Typos

Cambridge Technology Partners

Is there a way to fix poor commit messages?

```
$ git commit --amend
$ git rebase --i HEAD~X
$ git notes
```

Rollin' back

Objectives: Learn how to undo changes made in the repository.

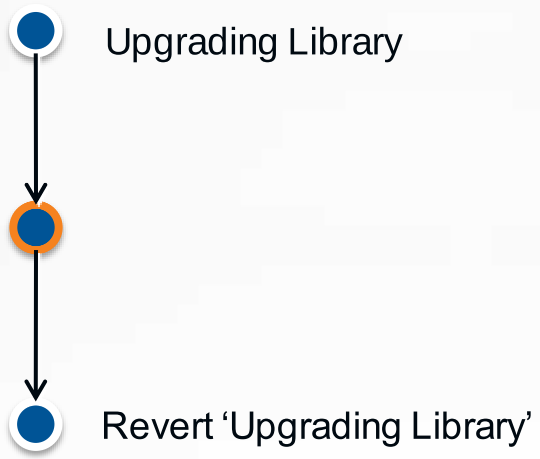
- Undo changes from different phases in git flow

```
$ git checkout --
$ git reset HEAD file
$ git reset HEAD^
```

Revert

```
$ git revert
```

git revert



One bridge too far

Disaster recovery.

What if…

```
$ git reset --hard HEAD^

$ git reflog
$ git reset --hard HEAD@{X}
```

Repeat Yourself
Repeat Yourself
Repeat Yourself

Reuse Recorded Resolution (ReReRe)

```
$ git config rerere.enabled true
… # create a merge conflict
$ git rerere status
$ git rerere diff
… # resolve conflict
$ git rerere diff
```

… # commit, reset hard HEAD^1, redo merge

git rerere

Evict old recorded resolutions from repository:

```
$ git rerere gc
```

Hooks

Git hooks

```
$ cd .git/hooks
```

Client-side

- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit

Server-side

- pre-receive
- post-receive
- update

Migration

SVN to Git migration

- Clone using git-svn (can take ages; make sure to properly map authors)
- Mirror currently used SVN repo to newly create Git repository
  - Run sync script every commit
- Move your infrastructure first (CI, code review tools etc.)
- Then move the team :)

Get IT right

Thank you!

Credits

- Icons provided by Icons8: http://icons8.com/