

Git

„The stupid content tracker”

Istvan Szekeres

11/03/2013

Background & History

- Linux kernel development: massively distributed, coordinated by 1 single guy (Linus Torvalds)
- Using nothing first (email..), then a proprietary system (BitKeeper)
- After some conflicts BK was given up
- Looking for alternatives: nothing found
- Requirements:
 - Email client integration
 - Massively distributed / decentralized
 - Fast. I mean FAST. (local operations < 1 sec)
 - Safe against data corruption / intrusion.
- April 2005: git was born (by Linus Torvalds)
 - Shell scripts on the surface (porcelain), tiny C binaries behind the scenes (plumbing)

Features

- Support for non-linear development
- Distributed development
- Compatibility with existing protocols (HTTP, FTP, rsync, ssh, email)
- Efficient handling of large projects
- Cryptographic authentication of history
- Toolkit-based design
- Pluggable merge strategies
- Repository optimizations
- Made for Linux, ported for Windows (cygwin)
- Import from Perforce, CVS/RCS, SVN
- Work with SVN

Hello, world – creating the repository

- Just create a working directory and initialize it

```
[5]% mkdir hello-world
```

```
[6]% cd hello-world
```

```
[7]% time git init
```

```
Initialized empty Git repository in /var/tmp/hello-world/.git/
```

```
git init 0.00s user 0.00s system 2% cpu 0.280 total
```

```
[8]% ls -la
```

```
total 24
```

```
drwxrwxr-x  3 szekeres szekeres 4096 Nov 16 12:13 .
```

```
drwxrwxrwt 60 root    root  16384 Nov 16 12:13 ..
```

```
drwxrwxr-x  7 szekeres szekeres 4096 Nov 16 12:13 .git
```

- git init - very fast repository initialization
- The repository is in .git

Hello, world – adding some files

- Create some files and check what Git thinks about them

```
[9]% echo "hello" > hello.txt
[10]% time git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    hello.txt
nothing added to commit but untracked files present (use "git add" to track)
git status 0.00s user 0.00s system 86% cpu 0.005 total
```

- git status – how is the working tree vs the repo?

Hello, world – adding some files

- Tell Git about some new content

```
[21]% time git add hello.txt
git add hello.txt 0.00s user 0.00s system 99% cpu 0.005 total
[22]% time git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   hello.txt
#
git status 0.00s user 0.00s system 109% cpu 0.004 total
```

- git add – tells git there is a new version of a file. DOES NOT tell that from now on this file is to be tracked!
- „Staging area” for the changes to be committed

Hello, world – committing the changes

- Record the changes

```
[26]% time git commit -m "My first commit"
[master (root-commit) 0586713] My first commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 hello.txt
git commit -m "My first commit" 0.00s user 0.00s system 109% cpu 0.005 total
[27]% time git status
# On branch master
nothing to commit (working directory clean)
git status 0.00s user 0.00s system 95% cpu 0.004 total
```

- git commit – commits the changes in the staging area
- Commit message: by convention: 1 line short description + empty line + many lines long description. 1st line used by many tools.

Hello, world – doing some more changes

- Change the contents

```
[39]% echo "hello, world!\nHello, sunshine\!" > hello.txt
```

```
[40]% git diff
```

```
diff --git a/hello.txt b/hello.txt
```

```
index ce01362..7c46bb1 100644
```

```
--- a/hello.txt
```

```
+++ b/hello.txt
```

```
@@ -1 +1,2 @@
```

```
-hello
```

```
+hello, world!
```

```
+Hello, sunshine!
```

- git diff – runs „diff” to show the changes against staging area
- ce01362 – object id of original file (later..)
- 7c46bb1 – object id of new file

Hello, world – what changes were made?

- Check the changelog

```
[61]% git log  
commit a29e14c54c8c01d16b6237d7404802918267d963  
Author: Istvan Szekeres <szekeres>  
Date: Tue Nov 16 12:50:50 2010 +0100
```

My second commit

```
commit 0586713b3a353246189d9a59a647728369561b6f  
Author: Istvan Szekeres <szekeres>  
Date: Tue Nov 16 12:35:44 2010 +0100
```

My first commit

```
[62]% git shortlog  
Istvan Szekeres (2):  
    My first commit  
    My second commit
```

Hello, world – inspecting the changes

- What was changed in a commit?

```
[64|128]% git show 0586713b
commit 0586713b3a353246189d9a59a647728369561b6f
Author: Istvan Szekeres <szekeres>
Date: Tue Nov 16 12:35:44 2010 +0100
```

My first commit

```
diff --git a/hello.txt b/hello.txt
new file mode 100644
index 0000000..ce01362
--- /dev/null
+++ b/hello.txt
@@ -0,0 +1 @@
+hello
```

- Commit Ids can be shortened (but must keep them unique)

Hello, world – reverting changes in the local repo

- Oops, we committed the wrong thing!

```
[67]% git reset 'HEAD^'
```

```
Unstaged changes after reset:
```

```
M    hello.txt
```

- HEAD – reference to the last commit of the current branch
- ^ - refers to the parent object of what it follows
- So HEAD^ is the commit before the last one in the current branch
- git reset – reset actual HEAD to the specified state, keeping the changes.
- **DO NOT DO git reset on already published changes!**

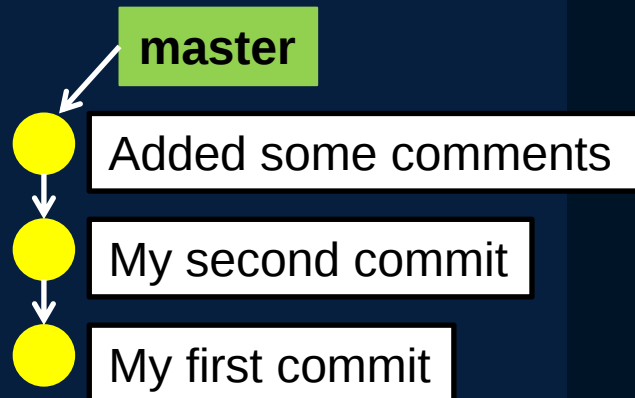
Rename tracking

- Git can figure out the changes...

```
[129]% cat hello.txt
hello, world!
Hello, sunshine!
[130]% echo 'Hello, trees!' >> hello.txt
[131]% mv hello.txt helloworld.txt
[132]% git add helloworld.txt
[134]% git rm hello.txt
rm 'hello.txt'
[135]% git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:   hello.txt -> helloworld.txt
#
[136]% git commit -m "renametest"
[master edc0fc8] renametest
1 files changed, 1 insertions(+), 0 deletions(-)
rename hello.txt => helloworld.txt (68%)
```

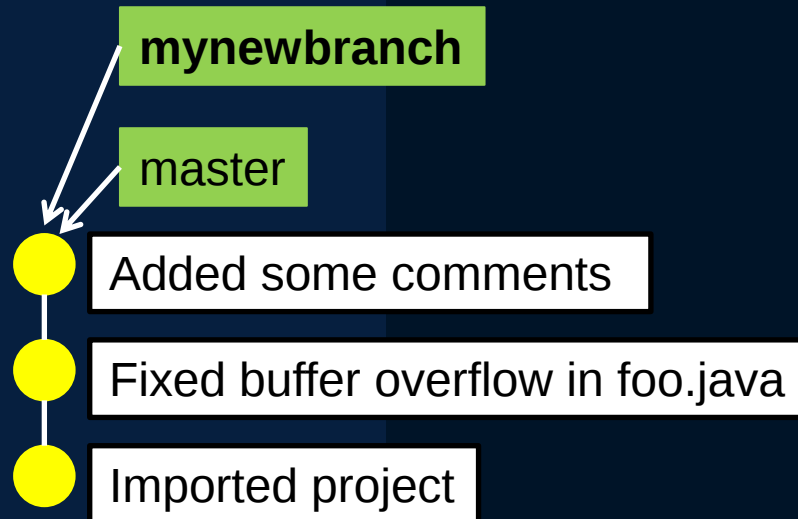
Branches - master

- Commits are like a linked list
- Each commit points to a/more parent(s)
- „master” is the name of the main development branch
- and technically „master” is a „pointer” to the commit that represents the head of the branch



Branches – creating a new one

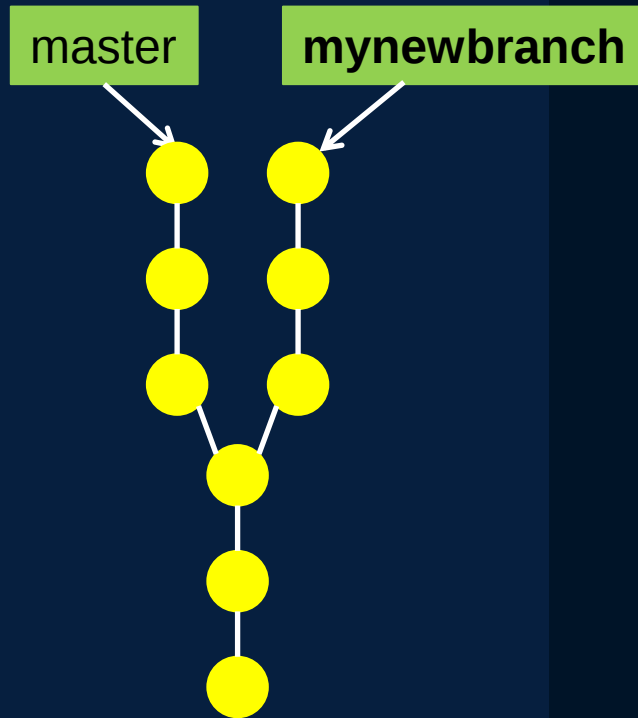
- `git branch mynewbranch` – creates a new branch at where HEAD points to
- `git checkout mynewbranch` – checks out content of mynewbranch



- What happens on branch creation? A new reference is made. Time required: ~0.
- What happens on checking out another branch: the files different in the two branches are updated. Time required: ~0.

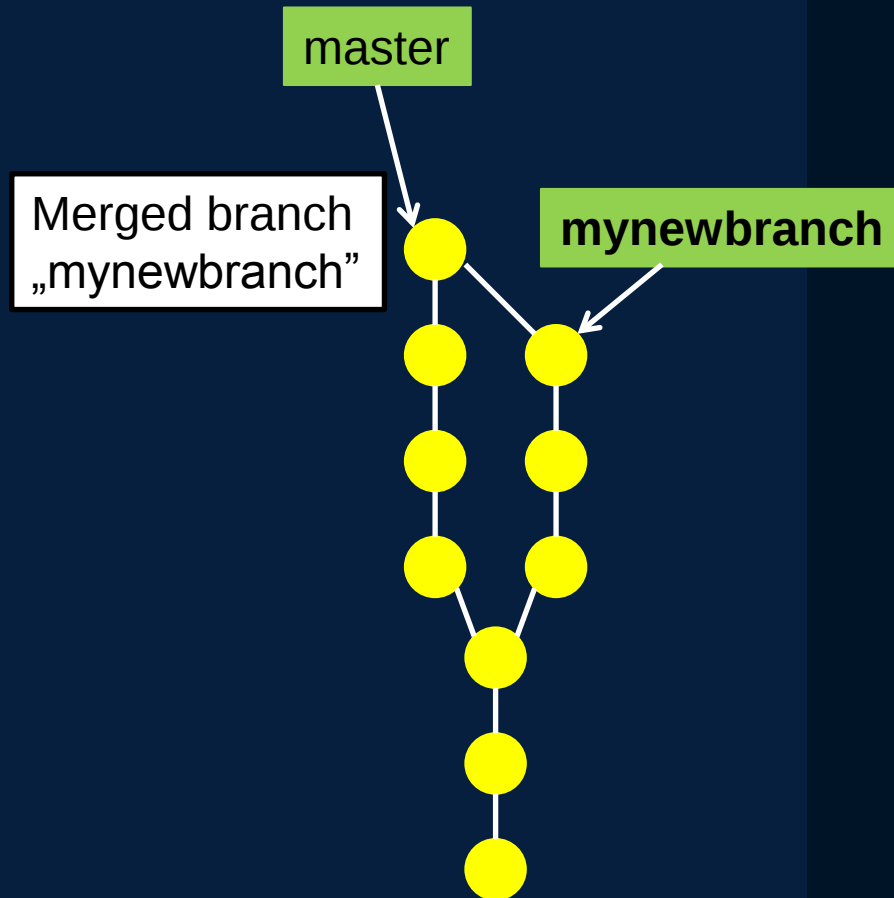
Branches – committing into branches

- `git branch mynewbranch` – creates a new branch at where HEAD points to
- `git checkout mynewbranch` – checks out content of mynewbranch

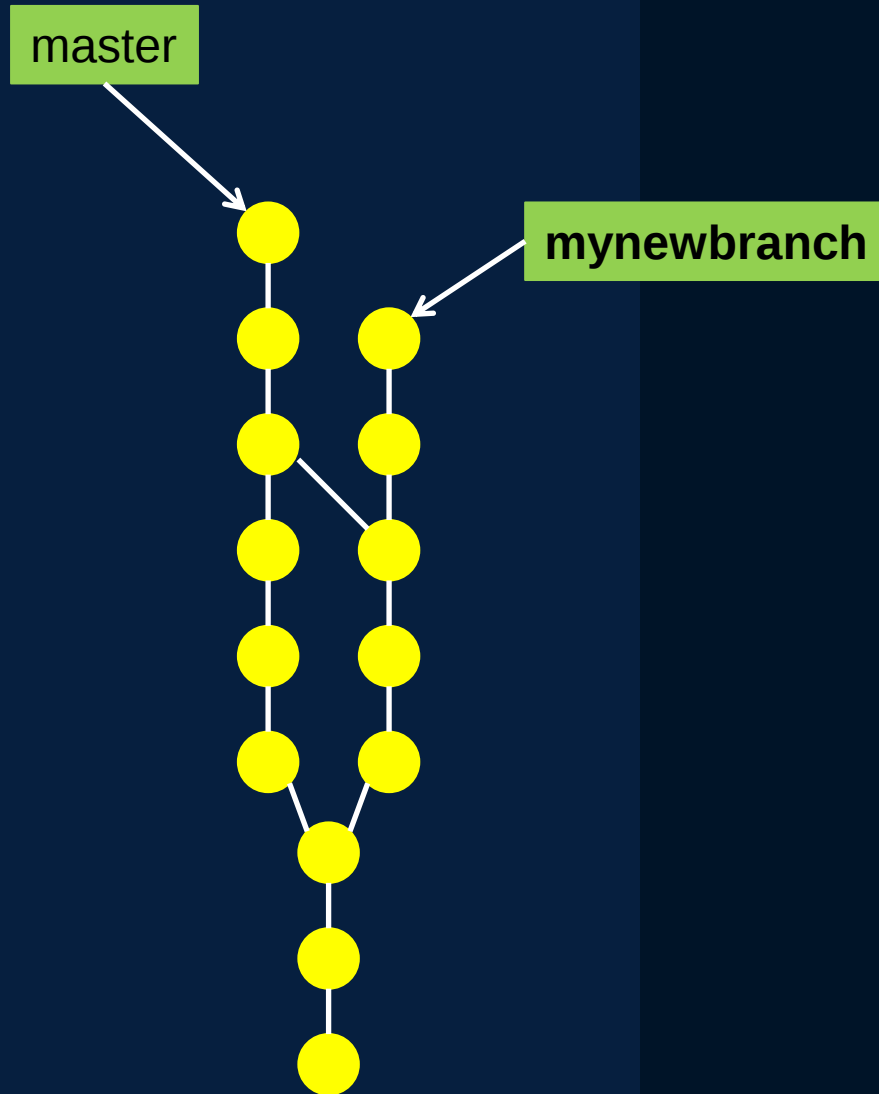


Branches – merging branches

- git checkout master – we want to merge into master
- git merge mynewbranch – checks out content of mynewbranch



Branches – and then the work can go on...



Remote branches

- git remote – where are other repositories? (local fs, http, git: ..)
- git fetch – retrieve objects (commits, etc) from other repositories

```
[113]% git remote -v
```

```
joseph /elte/user/j/jane/src/tools/.git (fetch)
```

```
joseph /elte/user/j/jane/src/tools/.git (push)
```

```
origin /elte/dev/web/tools/SCM/farmtools.git (fetch)
```

```
origin /elte/dev/web/tools/SCM/farmtools.git (push)
```

```
[114]% git branch -a
```

```
* master
```

```
remotes/jane/development
```

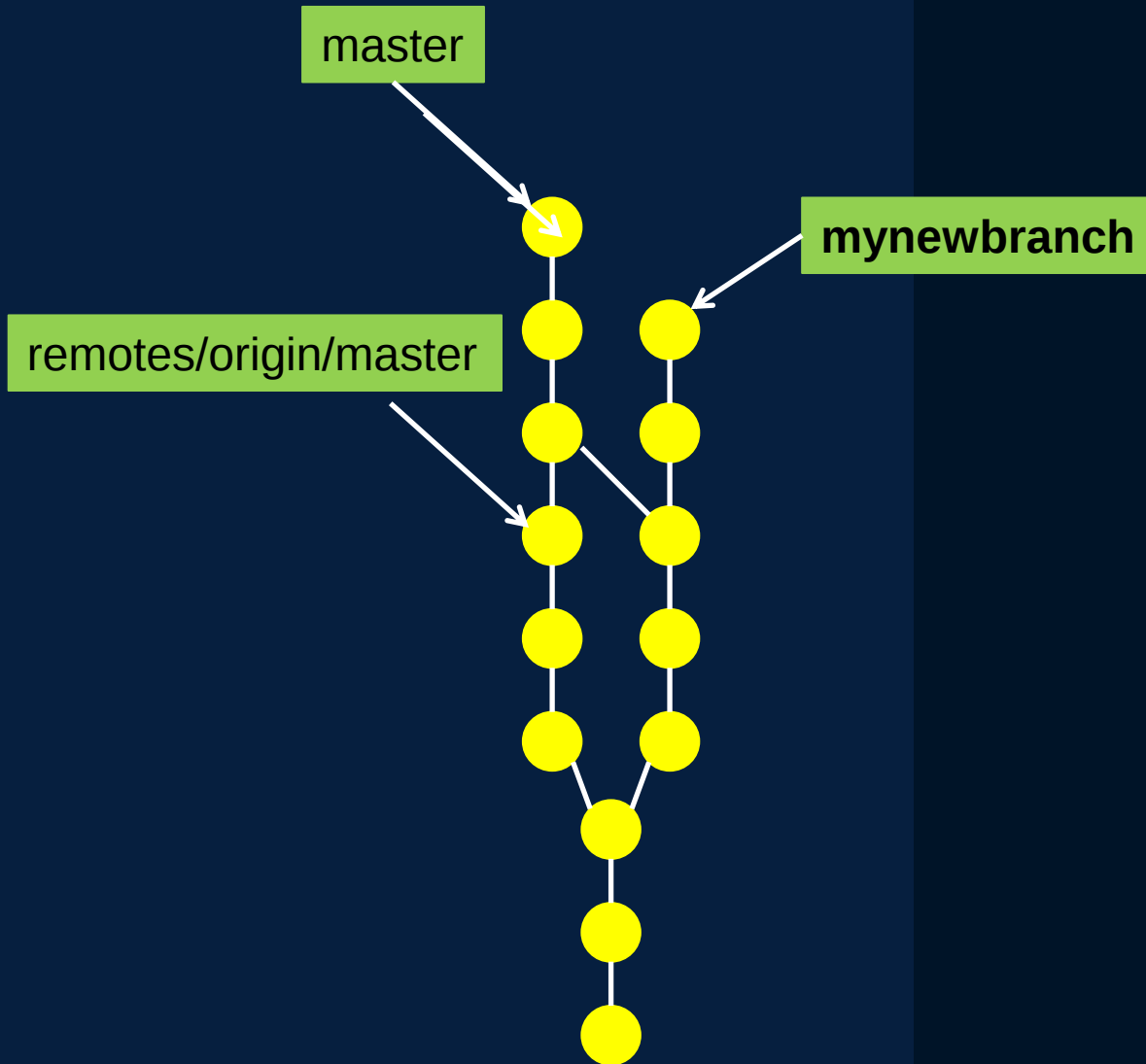
```
remotes/jane/master
```

```
remotes/origin/HEAD -> origin/master
```

```
remotes/origin/master
```

- Tracking branch – tracks what happens in remote branches

Remote branches - visualized



Remote branches – they are not that special!

- Local tracking branches are „mirrors” of remote branches (same IDs, etc.)
- `git fetch` – downloads changes from a remote repository into local tracking branch
- `git push` – upload local commits into remote repository
- Everything else is local !!!
- `git clone` – Clone remote repository
 - Sets up the git remote under the name „origin”
 - Fetches remote master branch (by default)
 - Sets up local master branch
- `git pull` - most common operation: fetch and merge remote changes
- „Bare” repositories – no working copy, just the repository

Working models

- Individual developer – Standalone
 - One repository, one developer
- Individual developer – Participant
 - One remote repository, more developers pushing directly into it
- Integrator
 - One dedicated repository maintained by an integrator
 - Integrator receives patches / pulls changes, then pushes into dedicated repository

Tags

- Shortcuts to specific commits
- „lightweight” tags – just references (like a symlink)
- „annotated” tags
 - actual committed objects
 - GPG-signable

```
[47]% git tag mytesttag dcaa01f2
```

```
[48]% git tag -l
```

```
mytesttag
```

```
[49]% git show mytesttag
```

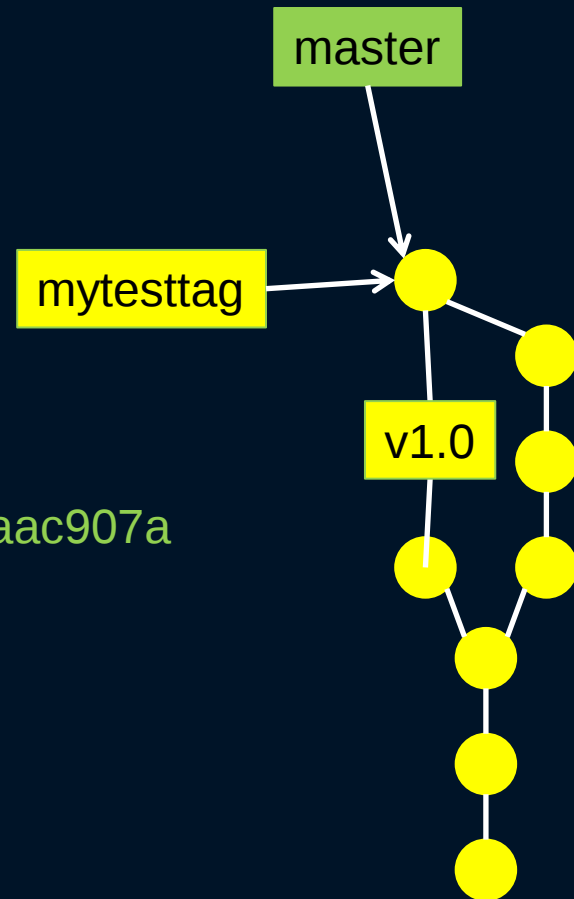
```
commit dcaa01f251871058d196dffc45373c7f5aac907a
```

```
Merge: d09fed7 7533413
```

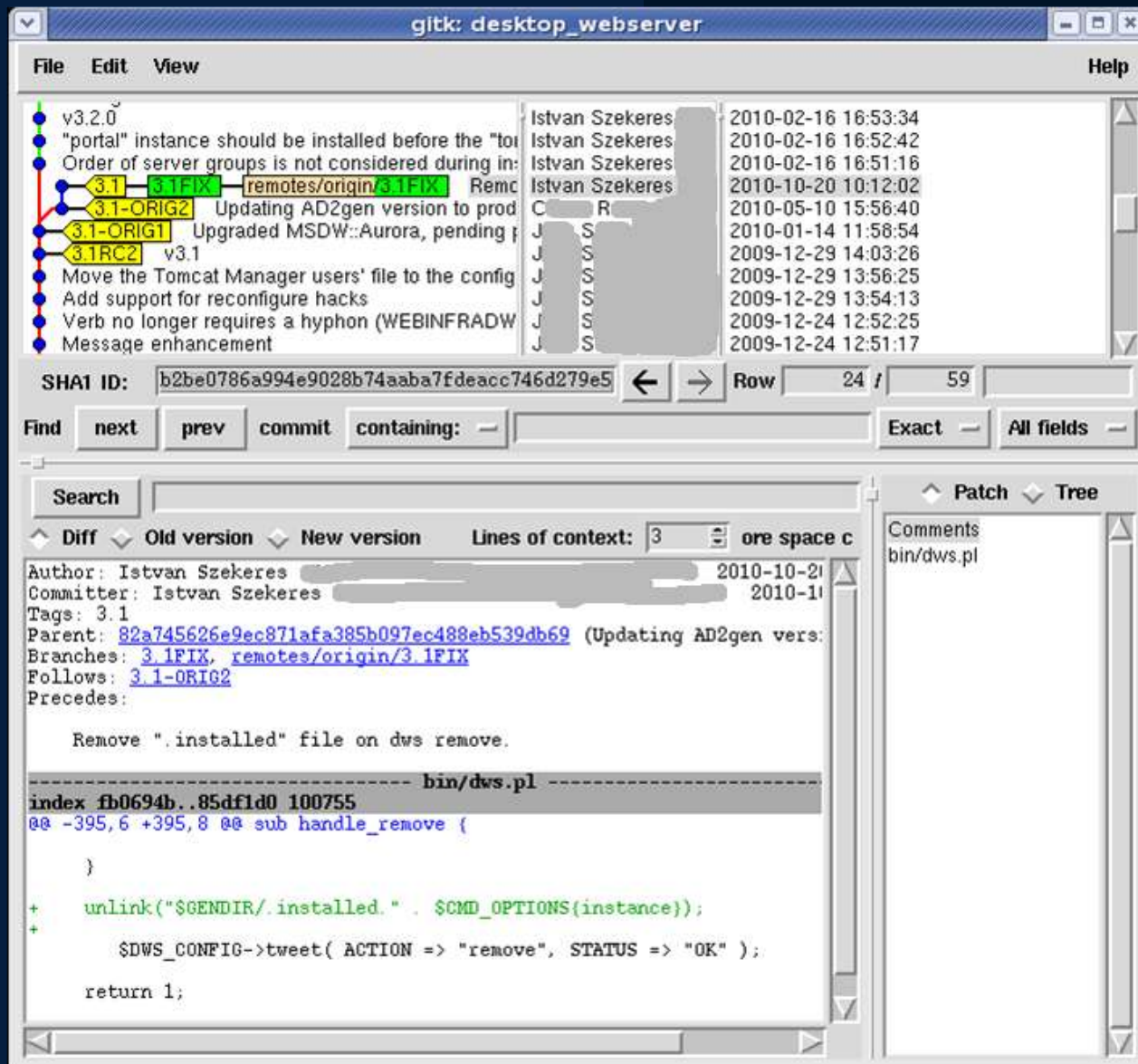
```
Author: Istvan Szekeres <szekeres>
```

```
Date: Tue Nov 16 13:54:45 2010 +0100
```

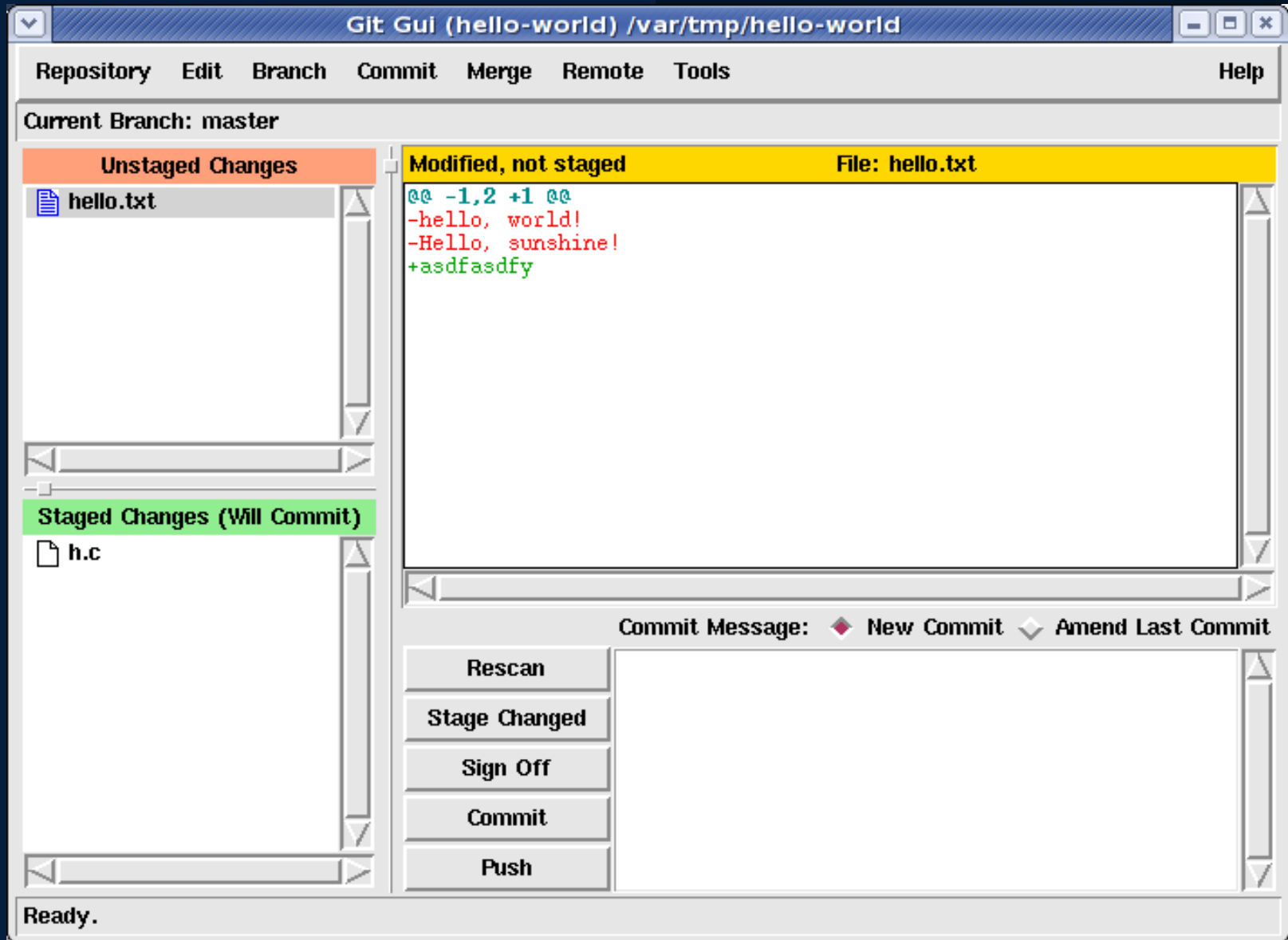
```
Merge branch 'mynewbranch'
```



GUI – gitk – Graphical history browser



GUI – git gui – Graphical commit tool



Objects

- All data are .. objects...
- objects are indexed by their SHA1 hash...
- ... which makes their ID unique (chance for collision is $1 / 2^{160}$)
- Revision ID (Integer, incremented by 1 on each commit) makes no sense

```
[14]% git cat-file -t 0586713
```

```
commit
```

```
[15]% git cat-file -p 0586713
```

```
tree aaa96ced2d9a1c8e72c56b253a0e2fe78393feb7
```

```
author Istvan Szekeres <szekeres> 1289907344 +0100
```

```
committer Istvan Szekeres <szekeres> 1289907344 +0100
```

My first commit

```
[16]% git cat-file -t aaa96ced2d9a1c8e72c56b253a0e2fe78393feb7
```

```
tree
```

```
[17]% git cat-file -p aaa96ced2d9a1c8e72c56b253a0e2fe78393feb7
```

```
100644 blob ce013625030ba8dba906f756967f9e9ca394464a    hello.txt
```

```
[18]% git cat-file -t ce013625030ba8dba906f756967f9e9ca394464a
```

```
blob
```

```
[19]% git cat-file -p ce013625030ba8dba906f756967f9e9ca394464a
```

```
hello
```

References

- All data are .. references...
- So,
 - where does our current branch (HEAD) points to?
 - where do branches point to?
 - where do tags point to?

```
[30]% cat .git/HEAD
```

```
ref: refs/heads/master
```

```
[31]% cat .git/refs/heads/master
```

```
dcaa01f251871058d196dffc45373c7f5aac907a
```

```
[32]% cat .git/refs/heads/mynewbranch
```

```
75334137342cd650ccf4997a422d3efc43b53616
```

```
[33]% cat .git/refs/tags/mytesttag
```

```
dcaa01f251871058d196dffc45373c7f5aac907a
```

Fun statistics #1

The git repository of git itself:

- tracks sources since **Apr 7, 2005**
- contains **23366 commits** (up to end of Sept 2010)
 - that is **~23 commits / day**
 - made by **~950 authors**
 - **~2000 files** in the latest version
- Working directory size (including sources, NOT including the repo itself):
17 MB
- Repository size:
69 MB

Fun statistics #2

The git repository of the Linux kernel (main motivation for creating git):

- tracks sources since **Apr 16, 2005**
- contains **~222000 commits** (up to mid of Nov 2010)
 - that is **~108 commits / day**
 - **~35000 files** in **~2160 directories** the latest version
- Working directory size (including sources, NOT including the repo itself):
468 MB
- Repository size:
400 MB

Links and further info

- On the internet
 - Git homepage
 - Pro Git
 - a very good Git book
 - Git for computer scientists