Getting started with the Git revision control system

Tim Retout

Rugby Linux User Group

14th April 2008

What is Git?

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

Installing Git

Debian `aptitude install git-core`

OpenSUSE `zypper install git-core`

Fedora `yum install git-core`

Gentoo `emerge -va git-core`

From source Get source from `http://git.or.cz/`, then:

```
$ ./configure
$ make
$ make install
```

MS-Windows Don't care. YDIW, etc.

Our first commit

What not to do at this point:

Our first commit

What not to do at this point:

```
$ git-<TAB>
Display all 137 possibilities? (y or n)
```

Our first commit

What not to do at this point:

```
$ git-<TAB>
Display all 137 possibilities? (y or n)
```

Don't panic!

Our first commit

What not to do at this point:

```
$ git-<TAB>
Display all 137 possibilities? (y or n)
```

Don't panic!

> *Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internal.*

Creating a repository

Creating a repository

```
$ git init
Initialized empty Git repository in .git/

$ ls -la
drwx------  3 lamby lamby 4096 2007-10-28 00:49 .
drwxr-xr-x 74 lamby lamby 4096 2007-10-28 00:49 ..
drwxr-xr-x  7 lamby lamby 4096 2007-10-28 00:49 .git
```

Your first commit

Committing is simple:

Your first commit

Committing is simple:

```
$ vim hello.py
$ git add hello.py

$ git commit
Created initial commit 0b7acf6: Intiial commit
1 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 hello.py
```

What just happened?

What just happened?

- Git has a staging area for commits

What just happened?

- Git has a staging area for commits
- `git add` adds files to the staging area
- `git commit` commits the staging area
- `git status` shows the status of the commit area

Reverting commits

Reverting commits

Reverting a commit (rollback)

```
$ git revert HEAD
```

This a seperate (but reverse) commit operation.

Reverting commits

Reverting a commit (rollback)

```
$ git revert HEAD
```

This a seperate (but reverse) commit operation.

Reverting changes

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

Resets working tree to last committed state.

Reverting commits

Reverting a commit (rollback)

```
$ git revert HEAD
```

This a seperate (but reverse) commit operation.

Reverting changes

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

Resets working tree to last committed state.

Fixing a commit

Run: `$ git commit -a --amend` after fixing broken files.

Sharing your work - overview

Sharing your work - overview

1. Create a new repository we can 'push' to
2. Configure our local repo to point to it
3. Push our changes
4. Let people know how to clone this repo!
5. Merging changes from other repositories

Creating a remote repository

- We need a repository somewhere to store our commits

Creating a remote repository

- We need a repository somewhere to store our commits
- ‘Bare‘ repositories vs. normal repositories

Creating a remote repository

- We need a repository somewhere to store our commits
- 'Bare' repositories vs. normal repositories

Creating a bare repository

```
$ mkdir myproject.git
$ cd !$
$ git --bare init
```

Configuring the local repository

- Each repo can 'track' other repositories, called *remotes*
- 'Remote' specifications are stored in `.git/config`

Configuring the local repository

- Each repo can 'track' other repositories, called *remotes*
- 'Remote' specifications are stored in `.git/config`
- Use `git remote` to add a new remote:

Tracking our remote repository

```
$ git remote add origin uwcs.co.uk:git/myproject.git
```

Pushing changes

- Once you have made some commits, you can push changes!

Pushing changes

- Once you have made some commits, you can push changes!
- Use `--all` the first time you push

Pushing changes (first time)

```
$ git push --all
```

Pushing changes

```
$ git push
```

Letting others clone your repository

Letting others clone your repository

- You can create your bare Git repository in `public_html` (or similar)

Letting others clone your repository

- You can create your bare Git repository in `public_html` (or similar)
- . . . or symlink a non-bare repository's `.git` into `public_html`

Letting others clone your repository

- You can create your bare Git repository in `public_html` (or similar)
- . . . or symlink a non-bare repository's `.git` into `public_html`
- . . . or use the Git protocol (recommended)

Letting others clone your repository

- You can create your bare Git repository in `public_html` (or similar)
- . . . or symlink a non-bare repository's `.git` into `public_html`
- . . . or use the Git protocol (recommended)

How others could clone your repository

```
$ git clone http://lamby.uwcs.co.uk/git/myproject.git
```

Letting others clone your repository

- You can create your bare Git repository in `public_html` (or similar)
- . . . or symlink a non-bare repository's `.git` into `public_html`
- . . . or use the Git protocol (recommended)

How others could clone your repository

```
$ git clone http://lamby.uwcs.co.uk/git/myproject.git
```

Using the 'dumb' protocols

If you are pushing to a repository that will be served over HTTP you must execute:

```
$ chmod +x hooks/post-update
```

Merging changes from others

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
```

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
$ git branch brad-branch
$ git checkout brad-branch
```

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
$ git branch brad-branch
$ git checkout brad-branch
$ git pull brad brads-new-cool-stuff
```

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
$ git branch brad-branch
$ git checkout brad-branch
$ git pull brad brads-new-cool-stuff
$ vim foo / git diff / ...
```

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
$ git branch brad-branch
$ git checkout brad-branch
$ git pull brad brads-new-cool-stuff
$ vim foo / git diff / ...
$ git checkout master
```

Merging changes from others

- Ask contributor to give you:
  1. His/her repository's URL (eg. `http://brad.uwcs.co.uk/git/myproject.git`)
  2. Which branch of his/her to pull (eg. `brads-new-cool-stuff`)

Merging changes from others

```
$ git remote add brad http://brad.uwcs.co.uk/git/myproject.git
$ git branch brad-branch
$ git checkout brad-branch
$ git pull brad brads-new-cool-stuff
$ vim foo / git diff / ...
$ git checkout master
$ git merge brad-stuff
```

Cool stuff

Speed

- Git is optimised for Linus:

Speed

- Git is optimised for Linus:
  - Fast merging
  - Fast branching

Speed

- Git is optimised for Linus:
  - Fast merging
  - Fast branching
  - Fast downloading

Speed

- Git is optimised for Linus:
  - Fast merging
  - Fast branching
  - Fast downloading
  - Laziness

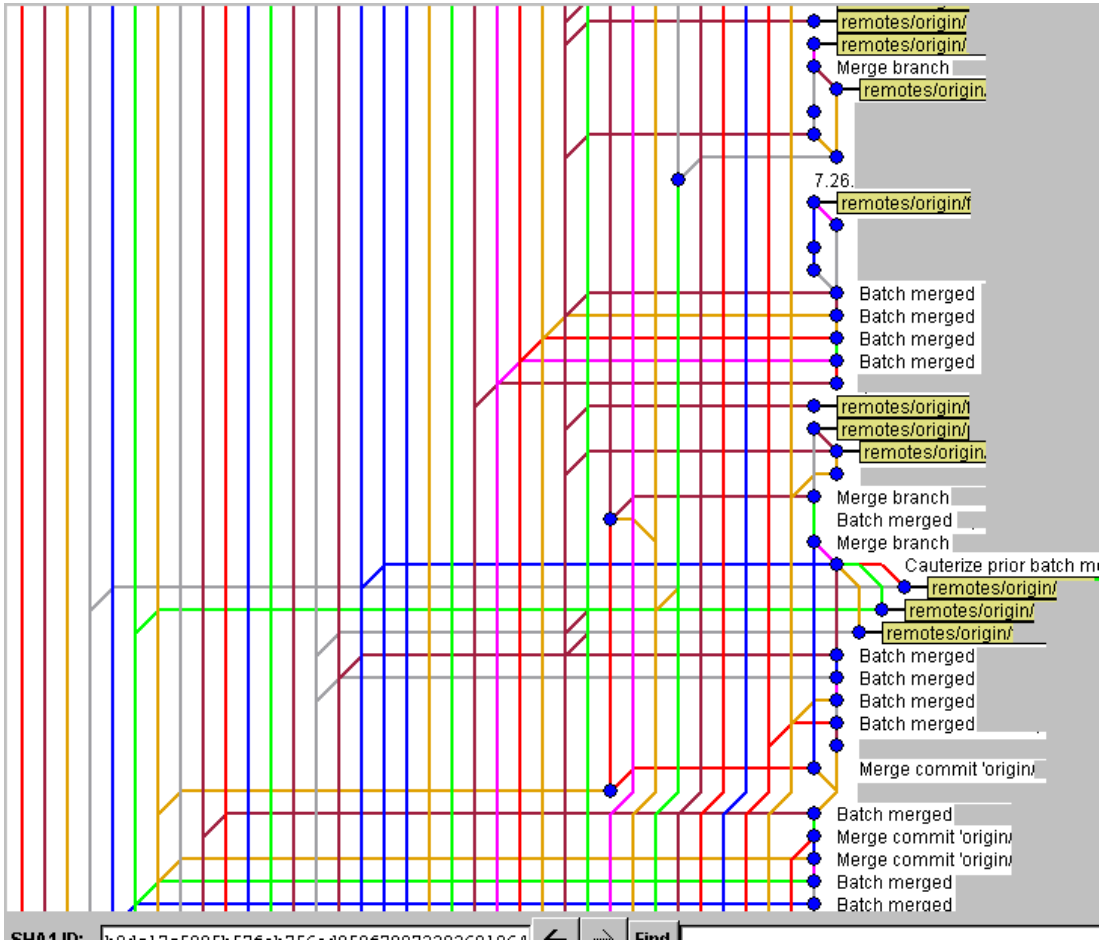
Speed

- Git is optimised for Linus:
  - Fast merging
  - Fast branching
  - Fast downloading
  - Laziness
- Garbage-collection: `git gc`

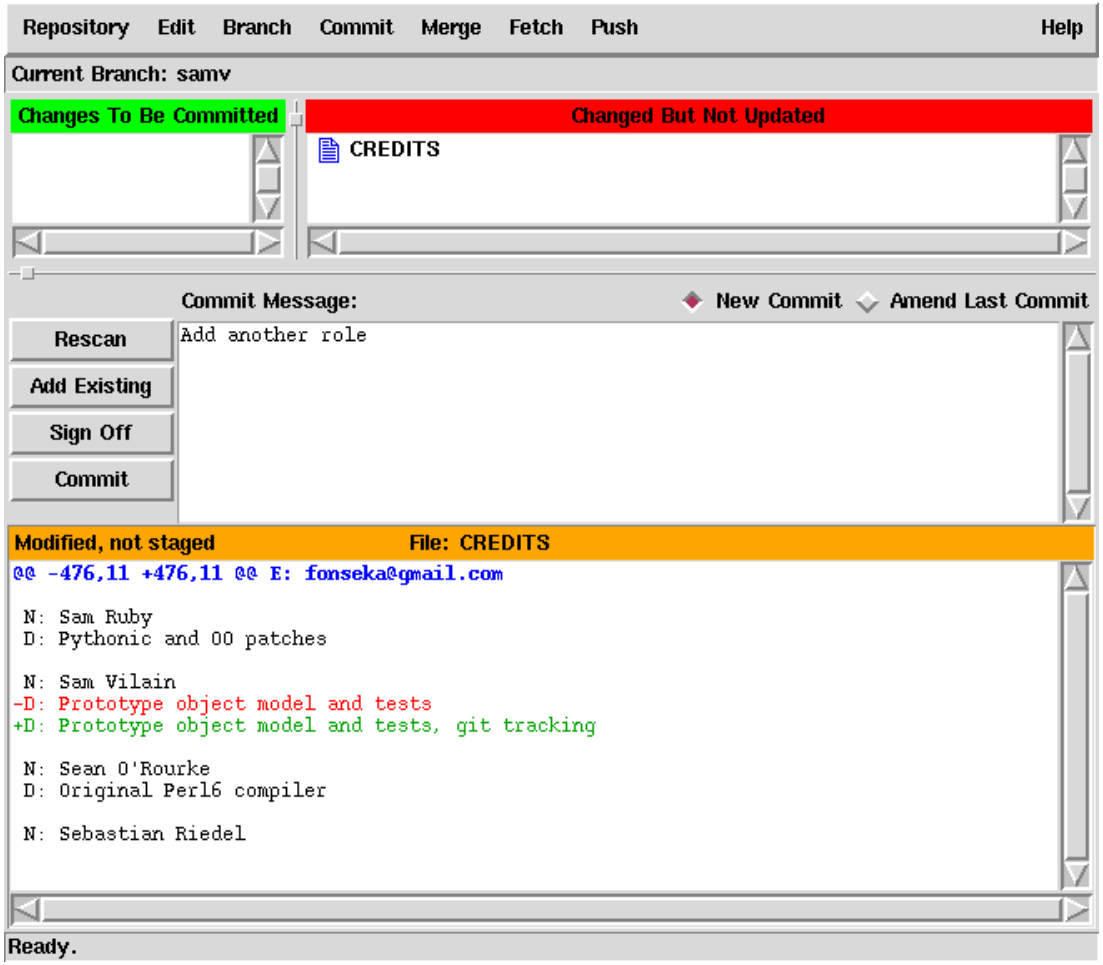
Speed

- Git is optimised for Linus:
  - Fast merging
  - Fast branching
  - Fast downloading
  - Laziness
- Garbage-collection: `git gc`
- Fast.. yet robust

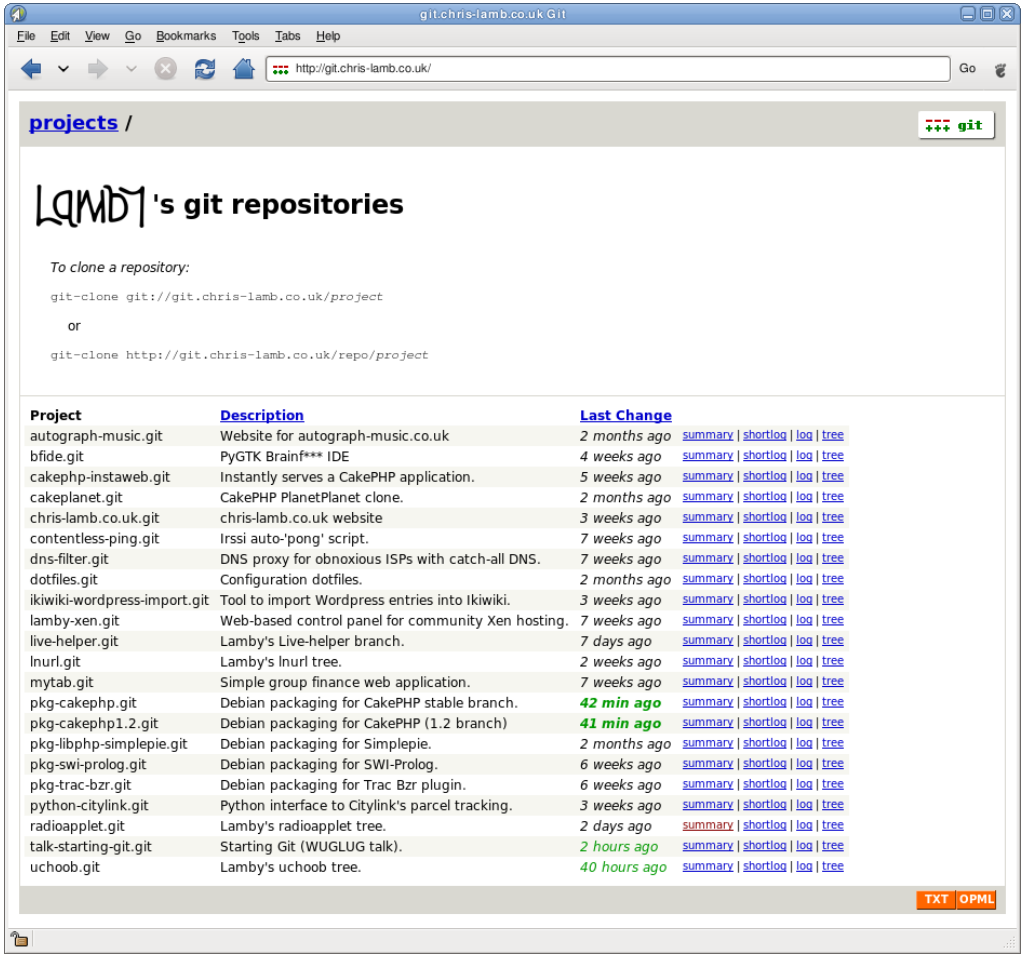
Gitk - 'the Git repository browser'



git-gui - ‘a portable graphical interface to Git’



Gitweb - ‘web interface to Git’



Importing from $TOOL

Importing from $TOOL

- Import complete history from:
  - Subversion / SVK
  - CVS
  - Perforce
  - Mercurial
  - Darcs
  - Arch, Quilt, IBM Rational ClearCase, ...

Importing from $TOOL

- Import complete history from:
  - Subversion / SVK
  - CVS
  - Perforce
  - Mercurial
  - Darcs
  - Arch, Quilt, IBM Rational ClearCase, ...
- Roll your own imports easily with `git-fast-import`.
- More info:
  `http://git.or.cz/gitwiki/InterfacesFrontendsAndTools`

Working transparently with SVN repositories

Working transparently with SVN repositories

- Get:
  - The familiar, faster and saner Git tools

Working transparently with SVN repositories

- Get:
  - The familiar, faster and saner Git tools
  - Off-line commits (whilst still being distributed)

Working transparently with SVN repositories

- Get:
  - The familiar, faster and saner Git tools
  - Off-line commits (whilst still being distributed)
  - A warm glow of elitism
- More info: `man git-svn`

‘Git-bisect’

- For quickly finding when a bug was introduced

‘Git-bisect’

- For quickly finding when a bug was introduced
- Binary searches history for fault

‘Git-bisect’

- For quickly finding when a bug was introduced
- Binary searches history for fault
- Works by either:
  - Manually specifying bad revisions (`git bisect good|bad`)

'Git-bisect'

- For quickly finding when a bug was introduced
- Binary searches history for fault
- Works by either:
  - Manually specifying bad revisions (`git bisect good|bad`)
  - Specifying a command that should be run on each revision

'Git-bisect'

- For quickly finding when a bug was introduced
- Binary searches history for fault
- Works by either:
  - Manually specifying bad revisions (`git bisect good|bad`)
  - Specifying a command that should be run on each revision
- Can be restricted to specified paths

'Git-bisect'

- For quickly finding when a bug was introduced
- Binary searches history for fault
- Works by either:
  - Manually specifying bad revisions (`git bisect good|bad`)
  - Specifying a command that should be run on each revision
- Can be restricted to specified paths
- Effective only with good patch discipline

‘Git-rebase’

- Update a local patch series against mainline

‘Git-rebase’

- Update a local patch series against mainline
- Good for developing local changes before sending upstream

‘Git-rebase’

- Update a local patch series against mainline
- Good for developing local changes before sending upstream
- Or move changesets from custom branches back onto to main line

Thanks!

RUGLUG contact information:

- Website: `http://rugby.lug.org.uk/`

Source for these slides is on the 'rugby' branch of:
`git://git.retout.co.uk/talk-starting-git.git`