VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Minttu Hämäläinen

Using Git

GitHub

Tekniikka ja liikenne
2015

SISÄLLYS

1 WHAT IS GIT

Git: Open source version control system, originally created for kernel development. Git is only a command line tool.

Git was original created for Linux, but works nowadays with most of mainly used operation systems as Windows, OS X, BSD, Solaris...

Author	created the work
Committer	last applied the work

Fetch	Getting latest version **to local** repository from remote repository
Push	Adding latest version **to remote** repository from local repository
Pull	Fetch and merge remote branch to your current branch

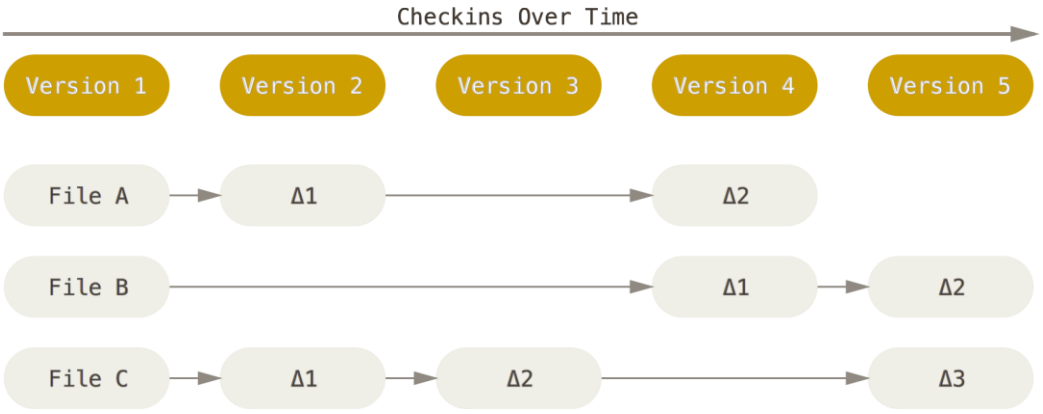
2 HOW GIT WOKRS?

Tämä luku kertoo mallipohjan käytöstä.

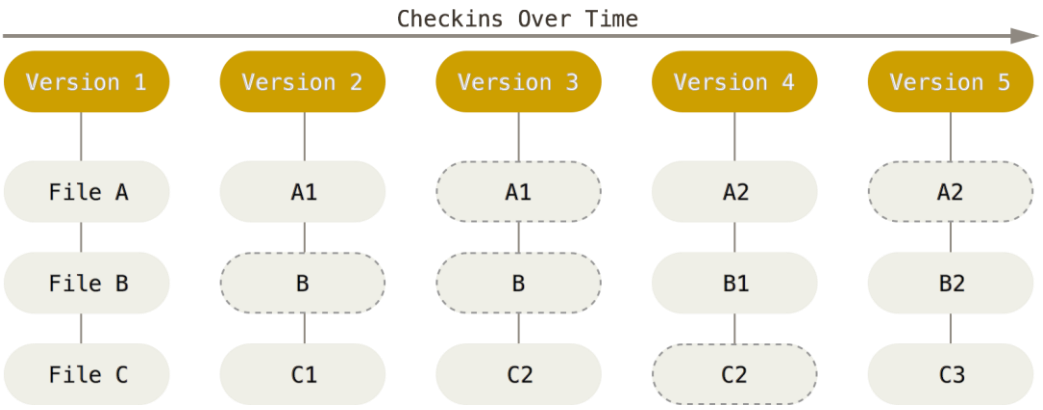
2.1 Data handling

Git's data handling is different than most other version control systems, because git saves a "snapshot" of what your all files look like every time when commit is made. If there is no difference in the previous file, git will just make a link to the previous file, not save the same file again.

Following pictures from 'http://git-scm.com/book/en/v2/Getting-Started-Git-Basics' will demonstrate the difference:



1. Storing data as changes to a base version of each file.



2. Storing data as snapshots of the project over time.

This is how user can go back or just pick the files needed:

User can

choose to use versions A1, B2 and C1

or

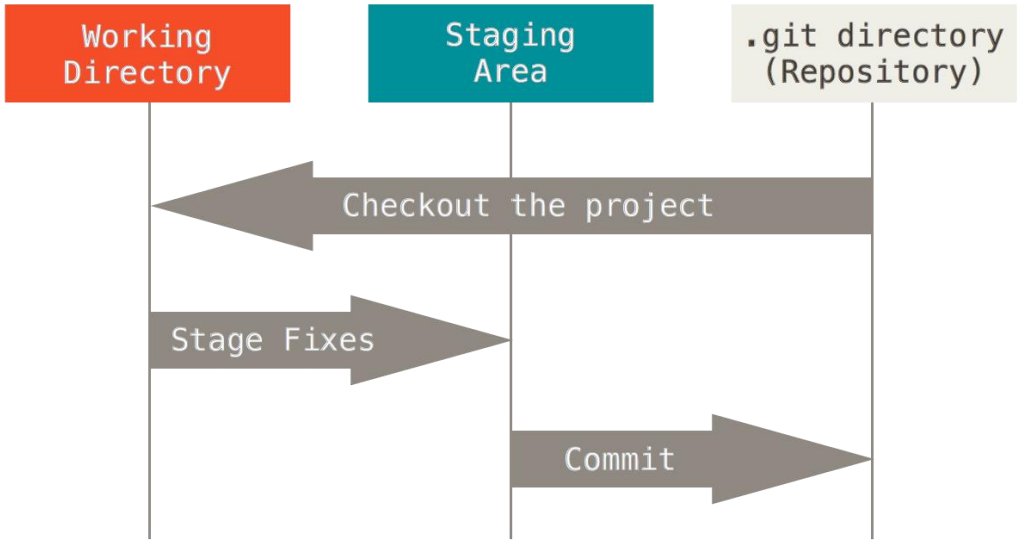
choose C1 and C3, and ignore changes made in commit C2. This is called "cherry picking"

2.2 Workflow

The basic Git workflow:

1. You **modify** files in your working directory.
2. You **stage** the files, adding snapshots of them to your staging area.
3. You do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Three main sections of Git project from 'http://git-scm.com/book/en/v2/Getting-Started-Git-Basics':



3. Working directory, staging area, and Git directory.

3 WHAT IS GITHUB?

GitHub: Git repository hosting service.

Basically it's just a website where you can store a copy of your own git repository. Git is the thing that does everything (GitHub related) that is happening in your end, in your computer.

It works with Windows and OS X.

GitHub Desktop client is available on https://desktop.github.com/ . With it you can use Git without command line usage.

3.1 Why to use GitHub?

GitHub adds its own features to the functionality of Git.

Graphical interface (desktop and mobile integration).

Private repositories and free accounts (public repositories).

User must have an account to create content to the site, but open source repositories are available for anyone to access.

4 HOW TO

4.1 Get Git

(a) Linux:

Use apt-get (Debian-based) or yum (like Fedora) or check *http://git-scm.com/download/linux* to install **git-all**

(b) Mac:

http://git-scm.com/download/mac (download newest version, works with all)

or

On Mavericks (10.9) or above you can do this simply by trying to **run *git* from the Terminal the very first time.**

If you don't have it installed already, it will prompt you to install it.

(c) Windows:

http://git-scm.com/download/win (download newest version, works with all)

(d) from Source (to get the most latest version):

You are going to need following libraries:

curl, zlib, openssl, expat, and libiconv

```
$ sudo yum install curl-devel expat-devel gettext-devel \
  openssl-devel perl-devel zlib-devel
$ sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \
  libz-dev libssl-dev
```

To add documentation in various formats:

```
$ sudo yum install asciidoc xmlto docbook2X
$ sudo apt-get install asciidoc xmlto docbook2x
```

Using Fedora/RHEL/RHEL-derivatives, you need to do this:

```
$ sudo ln -s /usr/bin/db2x_docbook2texi /usr/bin/docbook2x-texi
```

Grab the latest tagged release tarball: *https://github.com/git/git/releases*.

After that, compile and install:

```
$ tar -zxf git-2.0.0.tar.gz
$ cd git-2.0.0
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```

After this is done, you can also get Git via Git itself for updates:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

4.1.1 Setup

This needs to be done only once in the computer used.

First time customizing environment:

1. `/etc/gitconfig` file: Contains values for every user on the system and all their repositories. If you pass the option `--system` to `git config`, it reads and writes from this file specifically.

2. `~/.gitconfig` or `~/.config/git/config` file: Specific to your user. You can make Git read and write to this file specifically by passing the `--global` option.

3. `config` file in the Git directory (that is, `.git/config`) of whatever repository you're currently using: Specific to that single repository.

Each level overrides values in the previous level, so values in `.git/config` trump those in `/etc/gitconfig`.

On Windows systems, Git looks for the `.gitconfig` file in the `$HOME` directory (`C:\Users\$USER` for most people). It also still looks for `/etc/gitconfig`, although it's relative to the MSys root, which is wherever you decide to install Git on your Windows system when you run the installer. If you are using Git for

Windows 2.x or later, there is also a system-level config file at `C:\Documents and Settings\All Users\Application Data\Git\config` on Windows XP, and in `C:\ProgramData\Git\config` on Windows Vista and newer. This config file can only be changed by `git config -f <file>` as an admin.

source: http://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

---

4.1.2 Identity

Setting your username and email:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

If you want to override this with a different name or email address for specific projects, you can run the command without the `--global` option when you're in that project.

Using different name or email address for certain project, run command without `--global` –option while in that project

Source: http://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

4.1.3 Editor

Which editor is used when Git want's you to write in a message:

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

`On x86`

```
$ git config --global core.editor "'C:/Program
Files/Notepad++/notepad++.exe' -multiInst -nosession"
```

`On x64`

```
$ git config --global core.editor "'C:/Program Files
(x86)/Notepad++/notepad++.exe' -multiInst -nosession"
```

If no editor configured, Git will use the system's default editor

source: http://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

4.1.4 Settings

To check your settings, use:

```
$ git config --list
```

To check a specific key value, use:

```
$ git config user.name
```

4.1.5 Manpage

There is a manual page (manpage) help for all Git commands, and 3 ways to get to the manpage:

```
$ git help &lt;verb&gt;
$ git &lt;verb&gt; --help
$ man git-&lt;verb&gt;
```

Source: http://git-scm.com/book/en/v2/Getting-Started-Getting-Help

4.2 Use Git

4.2.1 Creating Repository

In project's directory create new subdirectory '.git':

```
$ git init
```

This only creates needed .git file, nothing is tracked yet!

4.2.2 Using Repository

Try this:

```
$ git status
```

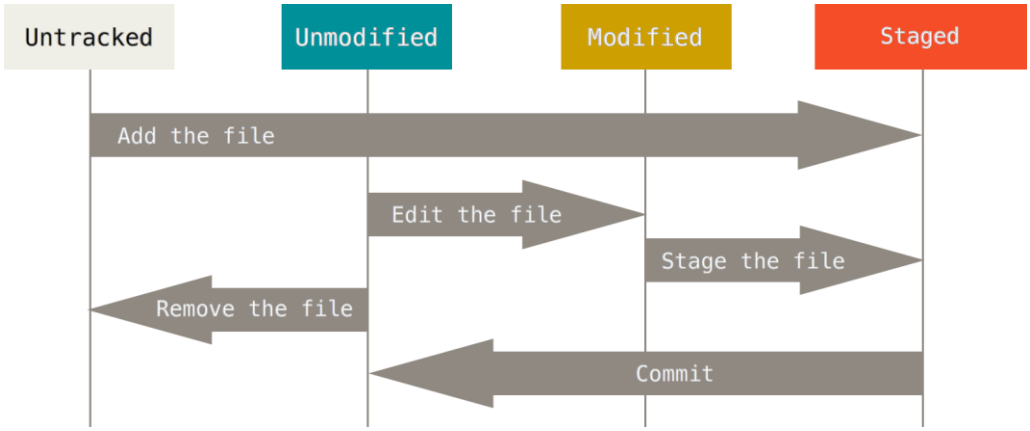
Your answer is:

```
On branch master
nothing to commit, working directory clean
```

Because! Even if there is files in the same directory/folder than created .git, without following there is nothing yet to be tracked.

Add and commit

```
$ git add *.c
$ git commit -m 'initial project version'
```



4. The lifecycle of the status of your files

```
$ git status
On branch master
Changes to be committed:

(use "git reset HEAD
&lt;file&gt;..." to un-
stage)

new file:   *.c
```

With

$ git add *.c

you make untracked or modified file '*.c' staged.

Git add –command adds chosen file to the Staged-state JUST THE WAY the file was when the command was given. If file is changed after, git add -command needs to be given again. Otherwise changed file will show under "`Changes not staged for commit:`" even though same file (earlier version) shows under "`Changes to be committed:`".

When all wanted files are staged:

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master 463dc4f] Story 182: Fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

If committing is done without parameter –m, will your editor ask for the message. If committing with parameter –v, you can see exactly which changes are you committing.

```
$ git commit -a -m 'added new benchmarks'
```

With parameter –a, commit will do the ‘git add‘-command to files already being tracked (all but “new” files).

NOTE: All files that have been committed are safe and can almost every time be recovered. Even files which are overwritten with ‘- -amend’ can be recovered. So anything lost without committing is most likely lost.

4.2.3 All little fun things

NOTE: Earlier used **git status** command can also be given as a “**short**”-version:

```
$ git status -s
 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```

Result: 2 columns, left for staged, right for modified:

- ?? = new files not tracked
- Ax = New files added to the staging area
- xM = modified
- MM = staged modifies and modifying which happened after being staged

With files you don’t want to be shown as untracked:

File named **.gitignore**

```
$ cat .gitignore
*.[oa]
*~
!lib.a
```

This will tell git to ignore files ending ‘.o’ ,‘.a’ and ‘~’ , exception beingn only file ‘lib.a’.

Bigger list found: *https://github.com/github/gitignore*

If you need to know exactly what was changed, not only which files, use

```
$ git diff
```

This will show you exact lines added and removed.

As **git diff** - - staged will show you what will go into next commit.

4.2.4 Move and remove

Moving files in git is easy, because git doesn’t really track file movement. So you can use commands ‘git mv’ to **rename** a file:

```
$ git mv file_from file_to
```

You can do the same thing with:

```
$ mv README.md README
$ git rm README.md
$ git add README
```

but really, first way is way shorter.

When **removing** file, you have to remove it from tracked files/staging area. After this you can commit. You can do that with `git rm`, which also removes the file from your working directory so next time committing it won't show as untracked.

```
$ git rm PROJECTS.md
rm 'PROJECTS.md'
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD &lt;file&gt;..." to unstage)

    deleted:    PROJECTS.md
```

If you remove the file **from your working directory**, you didn't get rid of it :

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm &lt;file&gt;..." to update what will be commit-
ted)
  (use "git checkout -- &lt;file&gt;..." to discard changes in
working directory)
deleted:    PROJECTS.md

no changes added to commit (use "git add" and/or "git commit -a")
```

If you want to keep the file in your working directory but don't want git to track it, you can use `--cached` option:

4.3 Use the commit history

When you want to look back to what's happened:

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon &lt;schacon@gee-mail.com&gt;
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon &lt;schacon@gee-mail.com&gt;
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon &lt;schacon@gee-mail.com&gt;
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

With command 'git log' will the history appear in reverse chronological order, first the recent. Most popular parameters used with git log are:

-p	Shows the difference between each commit
-2	Shows last 2 commits
--stat	Statistics of commits: how many file changed, how many lines added, summary
--since	git log –since=2.weeks
--pretty	Changes the default output

```
 $ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : changed the version number
085bb3b - Scott Chacon, 6 years ago : removed unnecessary test
a11bef0 - Scott Chacon, 6 years ago : first commit
```

More parameters listed:

http://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

4.4 Undo

4.4.1 Undo commit

What if you messed up your commit message or forgot to add some files? If you want to undo your commit:

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

This will use your staging area as your new commit. It's basically the same as normal commit, it just overwrites the last one you made.

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

Result: Just one pretty successful commit.

4.4.2 Undo staged file

```
$ git reset *.c
```

This command is shown when you run 'git status' –command:

```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD &lt;file&gt;..." to unstage)

    renamed:    README.md -&gt; README
    modified:   CONTRIBUTING.md
```

Command removes the file from staged area (modified, unstaged).

4.4.3 Unmodifying

When you have modified a file, but decided that you wanted last committed version back.

Again, git status saves the day!

```
Changes not staged for commit:
  (use "git add &lt;file&gt;..." to update what will be committed)
  (use "git checkout -- &lt;file&gt;..." to discard changes in
working directory)

    modified:   CONTRIBUTING.md
```

Using checkout:

```
$ git checkout -- CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD &lt;file&gt;..." to unstage)

    renamed:    README.md -&gt; README
```

This is dangerous command, you need to be sure. It copies over the other file. Changes are final with it.

Sources: http://git-scm.com/book/en/v2/Git-Basics-Undoing-Things

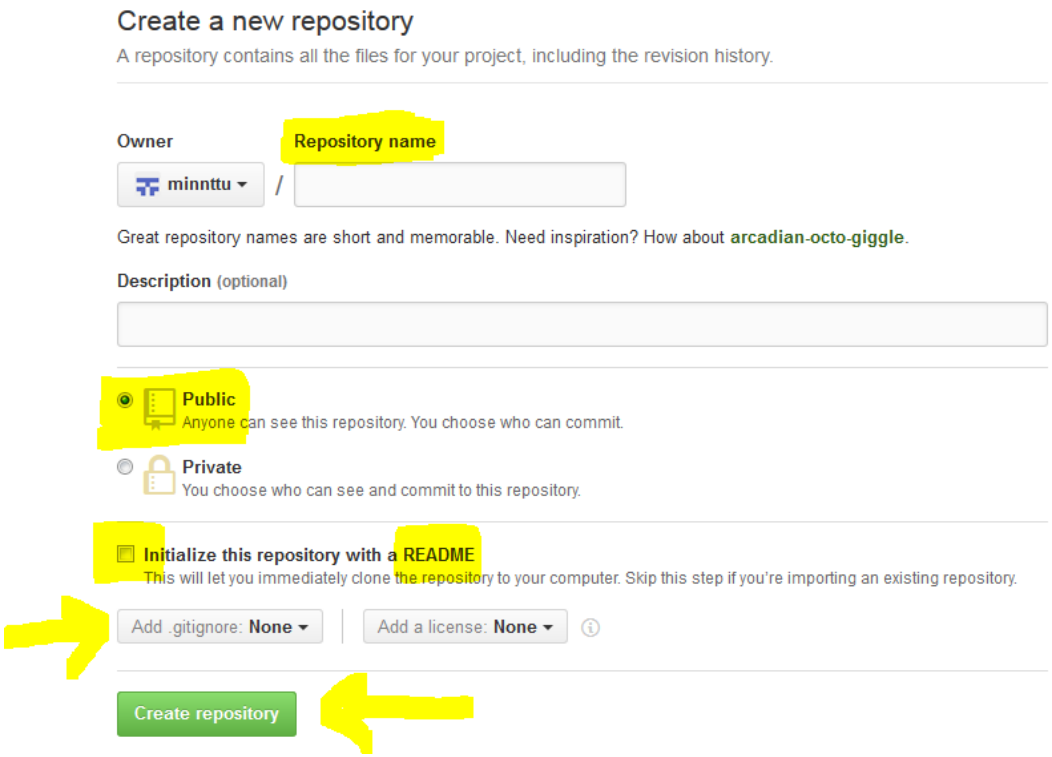
5 GITHUB

Working with others requires a remote repository. It is your repository just hosted somewhere in the internet.

Now we need a browser to create a user to GitHub. Free users' repository is available for anyone to access, main idea of open source coding.

After signing in, **you get url to your own remote repository.** Use it for pushing stuff from your local folder into the remote location

5.1 Creating a repository



You can choose if to add README, .gitignore or a license.

To locally find out which remote servers you can use: `$ git remote -v`

```
e1201352@shell:~/sovkeh/gameoflife$ git remote -v
origin  https://github.com/minnttu/GameOfLife.git (fetch)
origin  https://github.com/minnttu/GameOfLife.git (push)
```

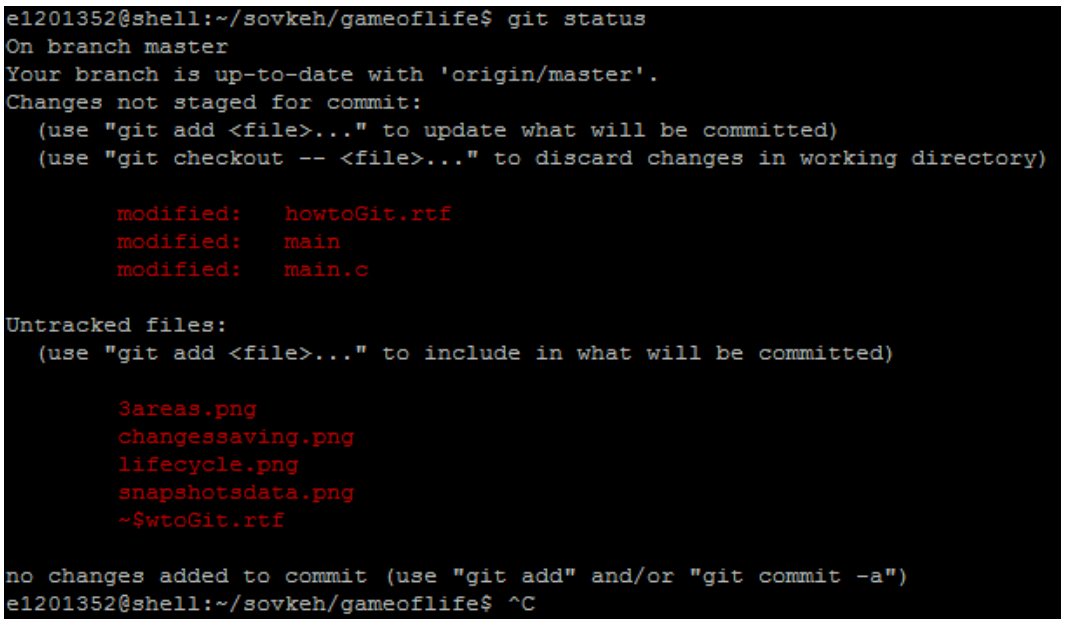
If you have the url for the repository you want to use:

```
git remote add [shortname] [url]
```

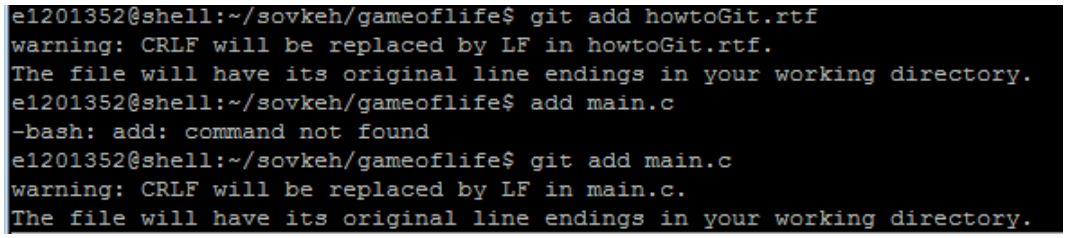
After this you don't need the whole url, you can use the shortname to fetch and push.

Repository will automatically add remote under the name 'origin'.

5.2 Example



5. $ git status



6. $ git add …

```
e1201352@shell:~/sovkeh/gameoflife$ git commit -m "how to almost ready, missing only
 adding a new remote repository and final how to's. Main should be about the same"
warning: CRLF will be replaced by LF in main.c.
The file will have its original line endings in your working directory.
[master a15e752] how to almost ready, missing only adding a new remote repository an
d final how to's. Main should be about the same
warning: CRLF will be replaced by LF in main.c.
The file will have its original line endings in your working directory.
 2 files changed, 51850 insertions(+), 947 deletions(-)
```

7. $ git commit

```
e1201352@shell:~/sovkeh/gameoflife$ git push origin
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Username for 'https://github.com': minnttu
Password for 'https://minnttu@github.com':
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 190.90 KiB | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://github.com/minnttu/GameOfLife.git
   53dbcef..a15e752  master -> master
e1201352@shell:~/sovkeh/gameoflife$
```

8. $ git push

LÄHTEET

http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

http://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

http://git-scm.com/book/en/v2/Getting-Started-Getting-Help

http://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

http://git-scm.com/book/en/v2/Git-Basics-Undoing-Things

http://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes

http://github.com