

Version Control and Issue Tracking (with thanks to James Hetherington, UCL RC)

- Managing code inventory
 - “When did I introduce this bug?”
 - Undoing mistakes
- Working with other programmers
 - How can I merge my work with Jim’s?
- What’s the most important bug to fix next?

What is version control? (Solo version)

- Do some programming
- `> my_vcs commit`
- Program some more
 - Realise mistake
- `> my_vcs rollback`
 - Mistake is undone

Syntax here is example only!

What is version control? (team version)

Sue

- Create some code
- `> my_vcs commit`
- ...wait...
- ...wait...
- ...wait...
- ...wait...
- `>my_vcs update`
- Do some programming
- ... program some more
- `> my_vcs commit`
 - Oh Noes! Error message!
- `> my_vcs update`
- `> my_vcs merge`
- `> my_vcs commit`
- More programming...

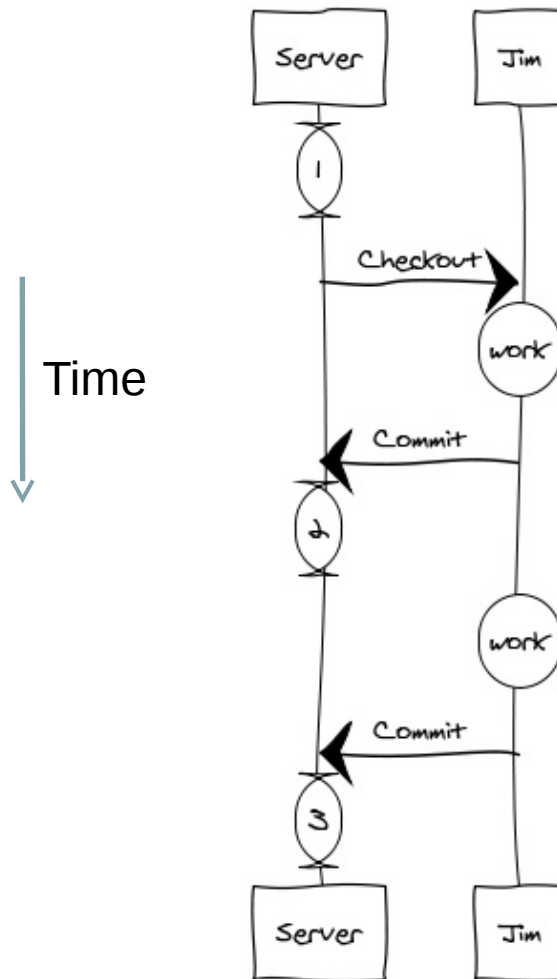
Jim

- ... wait ...
- ... wait ...
- Join the team
- `> my_vcs checkout`
- do some programming
- `> my_vcs commit`
- Do some programming
- ... more programming...
- `> my_vcs commit`
- ... more programming ...
- ... more programming ...
- ... more programming ...
- ... more programming ...
- ... more programming ...
- `> my_vcs commit`
 - Error again...

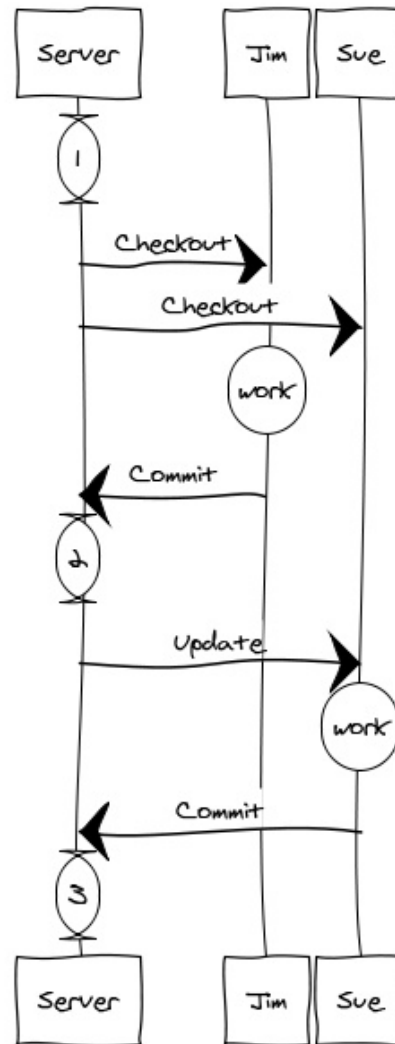
Centralised VCS concepts

- There is one, linear history of changes on the server or **repository**
 - Each revision has a unique identifier
- You have a **working copy**
- You **update** the working copy to match the state of the repository
- You **commit** your changes to the repository
- If you someone else has changed it you have to **resolve conflicts** between your changes and the repository, and then commit

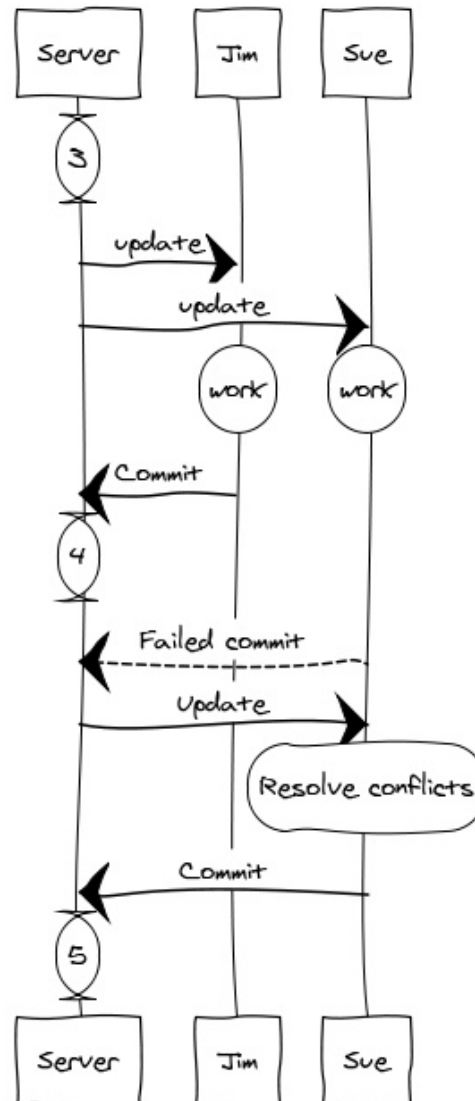
Centralised VCS solo workflow



Centralised VCS Team workflow: no conflicts



Centralised VCS with conflicts



Distributed and Centralized Version Control

- Centralized:
 - Some server contains the remote version
 - Your computer has your copy
 - To switch back to an old copy you need the internet
 - E.g. cvs, subversion (**svn**)
- Distributed:
 - Every user has a version of the full history
 - Users can synchronize their history with each other
 - Having a central “master” copy is a policy option
 - Most groups do this
 - E.g. **git**, mercurial (hg), bazaar (bZR)

Pragmatic distributed VCS

Git

```
git clone git@github.com:ucl/mycode.git
```

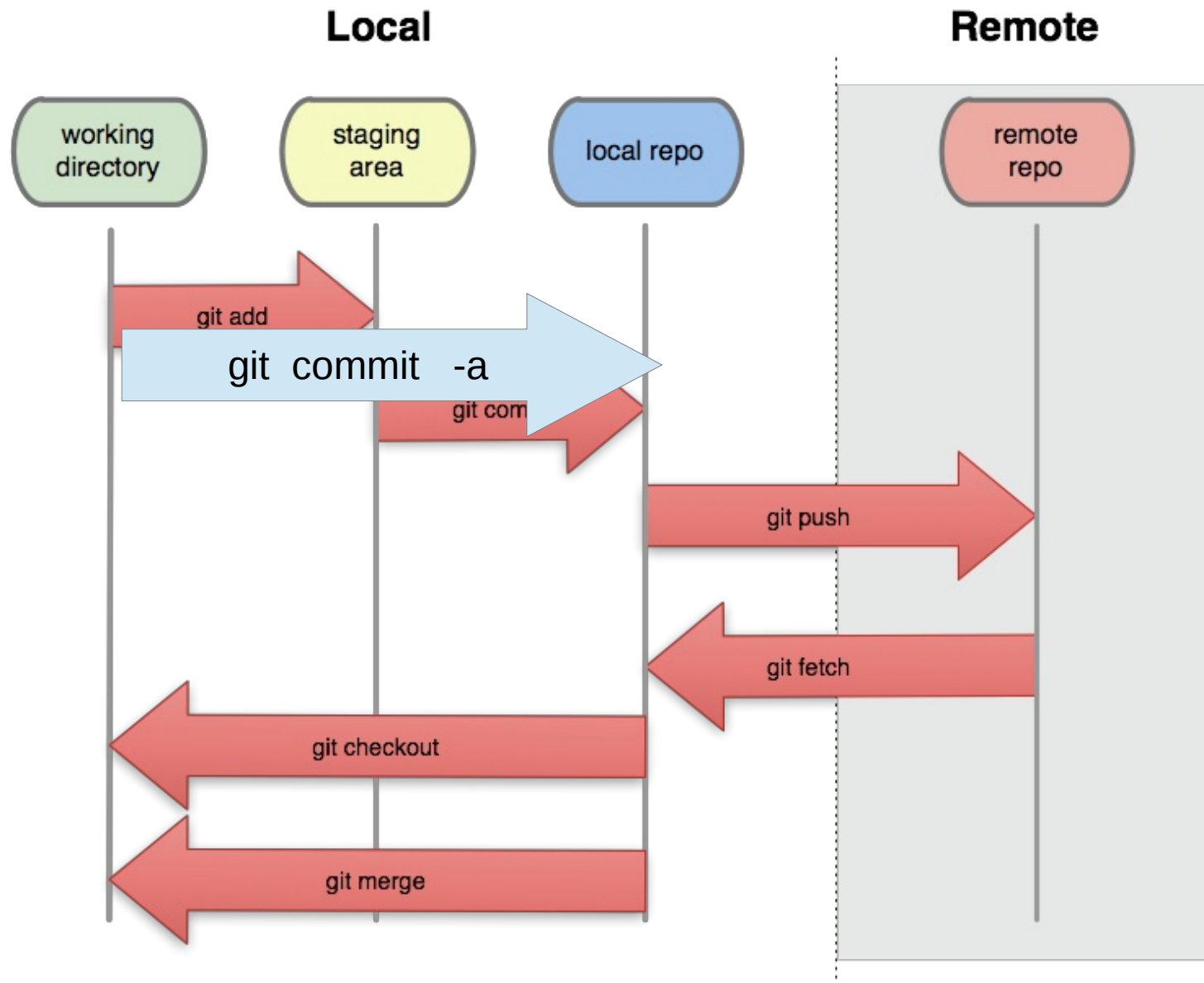
```
git commit -a
```

```
git push
```

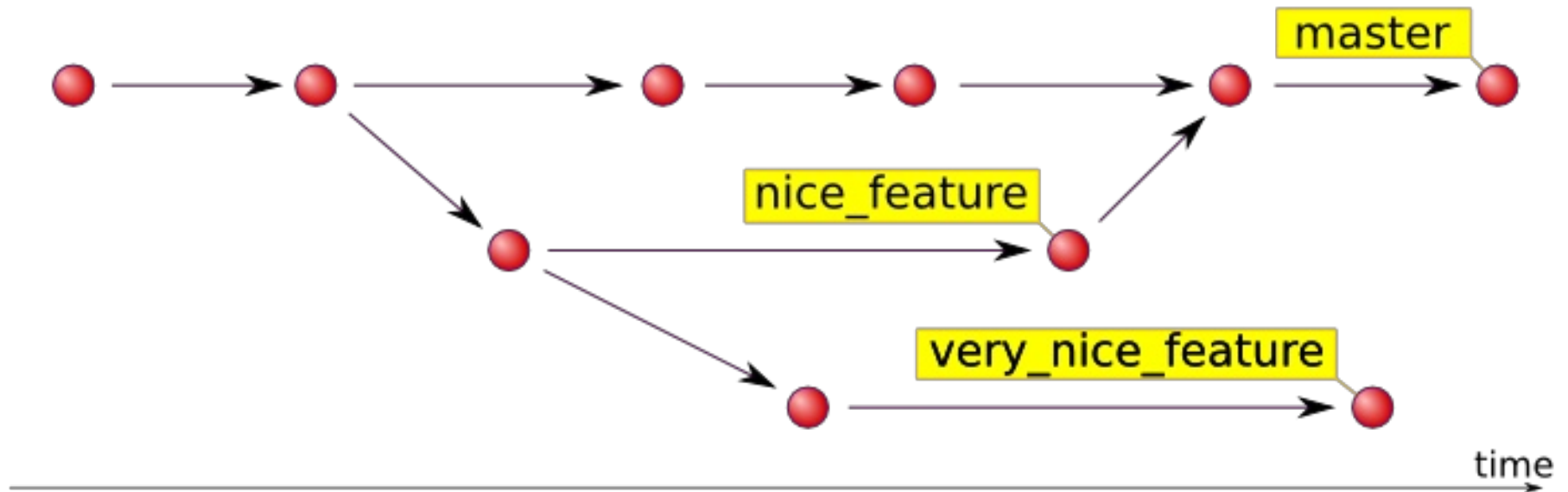
```
git pull
```

```
git status
```

```
git diff
```



Working with branches



Working with branches in git

```
> git branch
* master

> git checkout -b experiment

> git branch
master
* experiment
```

Sharing branches in git

```
git push origin experiment
```

publish the branch to remote

```
git push -u origin experiment
```

publish the branch to remote(first time)

```
git checkout origin/experiment
```

get a new branch from a remote

Merging and deleting branches

`git checkout master`

switch back to master branch

`git merge experiment`

take all the changes from experiment into master
exactly like merging someone else's work

`git branch -d experiment`

the experiment is done, get rid of local branch

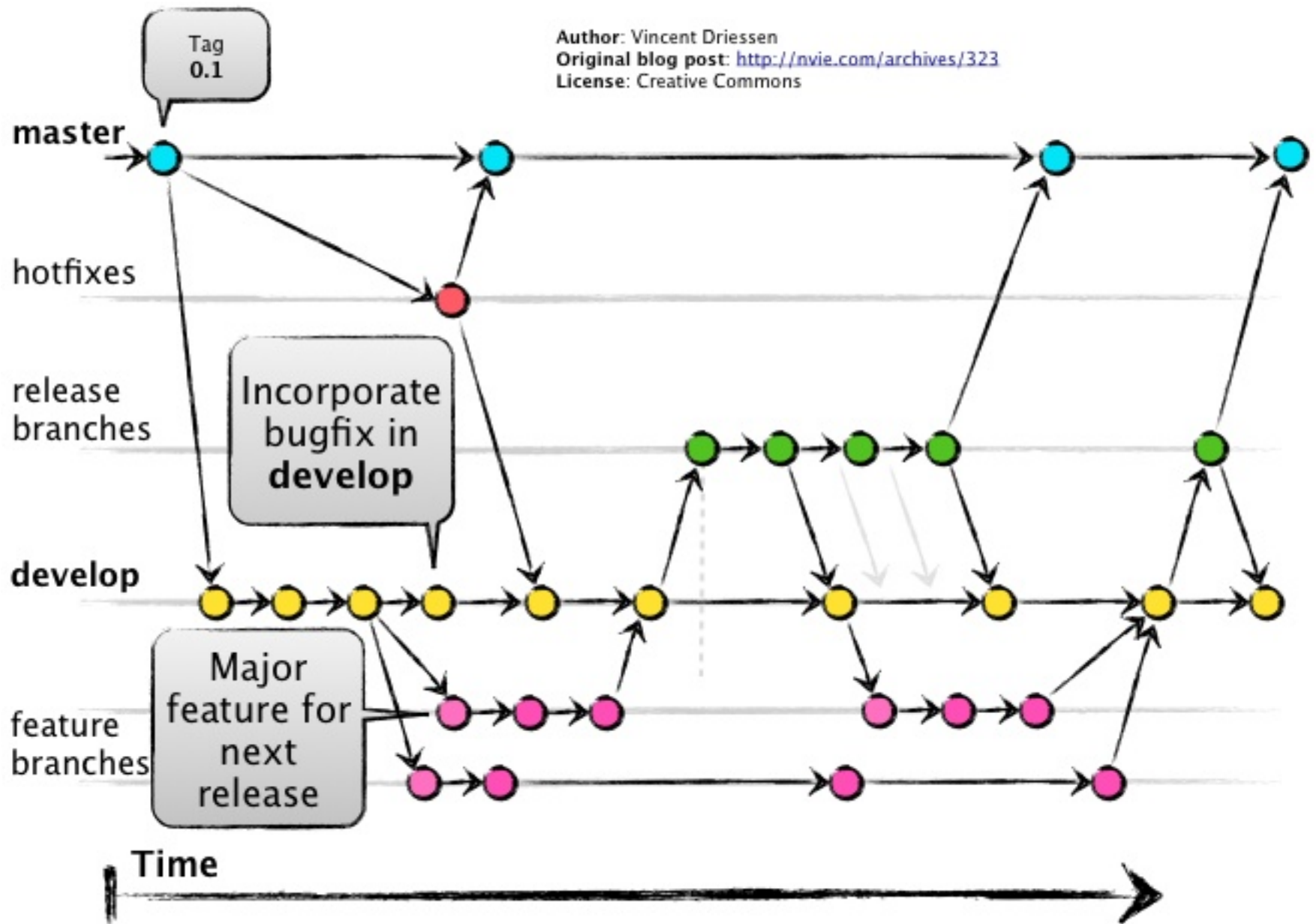
`git push --delete experiment`

git rid of the branch on the remote

Working with branches

- You should have a development branch and a stable branch
- You should create temporary branches for experimental changes
- If you release code to others, you should make a release branch
 - Then you can make fixes to bugs they find
 - And control which of your work goes in the release

Author: Vincent Driessen
 Original blog post: <http://nvie.com/archives/323>
 License: Creative Commons



Tagging

- You should tag working versions
- You should produce real science only with specific tagged versions, and note which one

Tagging

```
git tag v1.3
```

add a tag, labelling last commit

```
git tag v1.3 ab48dc
```

tag an old commit

```
git push --tags
```

publish the tags to origin

Working with GitHub

GitHub Bootcamp If you are still new to things, we've provided a few walkthroughs to get you started.



Set Up Git

A quick guide to help you get started with Git.

1



Create A Repository

Create the place where your commits will be stored.

2



Fork a Repository

Copy a repo to create a new, unique project from its contents.

3



Be social

Follow a friend.
Watch a project.

4

Set up ssh keys

SSH Keys		Add SSH key
jamespjh@plinian	(ce:f6:e2:64:22:15:75:66:43:f5:66:5b:63:18:f6:37)	Delete
GitHub for Mac - szilard	(0d:f3:58:37:b9:a7:3d:50:a8:c1:b7:50:51:41:3b:93)	Delete
jamespjh@szilard	(d1:81:7e:58:16:30:00:50:33:1e:76:f1:57:29:dd:39)	Delete


Add an SSH Key


Title

Key

Add key



Create repository

PUBLIC 

Owner  **jamespjh** ▾ /

Great repository names are short and memorable. Need inspiration? How about **cloaked-nemesis**.

Description (optional)

- ☒  **Public**
 Anyone can see this repository. You choose who can commit.
- ☐  **Private**
 You choose who can see and commit to this repository.

- ☐ **Initialize this repository with a README**
 This will allow you to `git clone` the repository immediately.

Add .gitignore: **None** ▾

Create repository

Conclusions

- Tools can make your development easier, safer, more reliable, more correct, and more collaborative
- They can be complicated and take time to learn
- Learn by practicing
 - Use the tools
 - Pick an open source project on github or bitbucket and start contributing

<http://git-scm.com/book/>

<http://svnbook.red-bean.com/>