Git

David Hubber
Code Coffee
13th March 2018

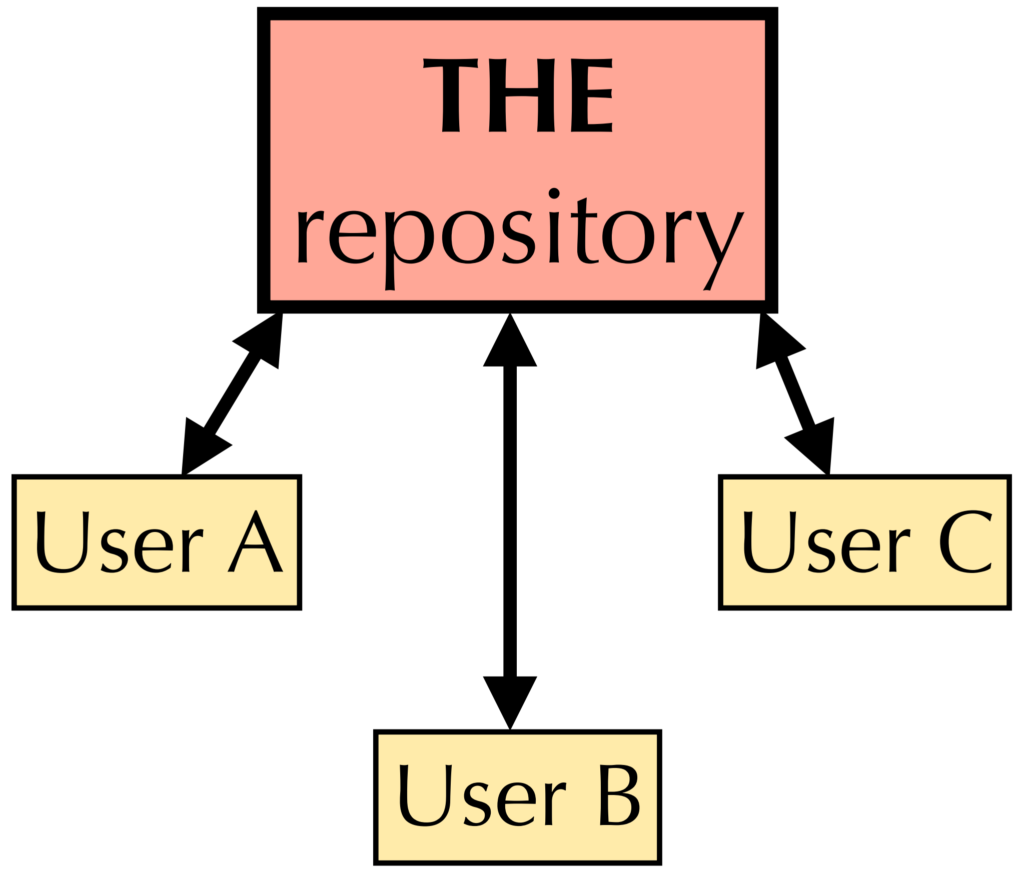
A few questions …

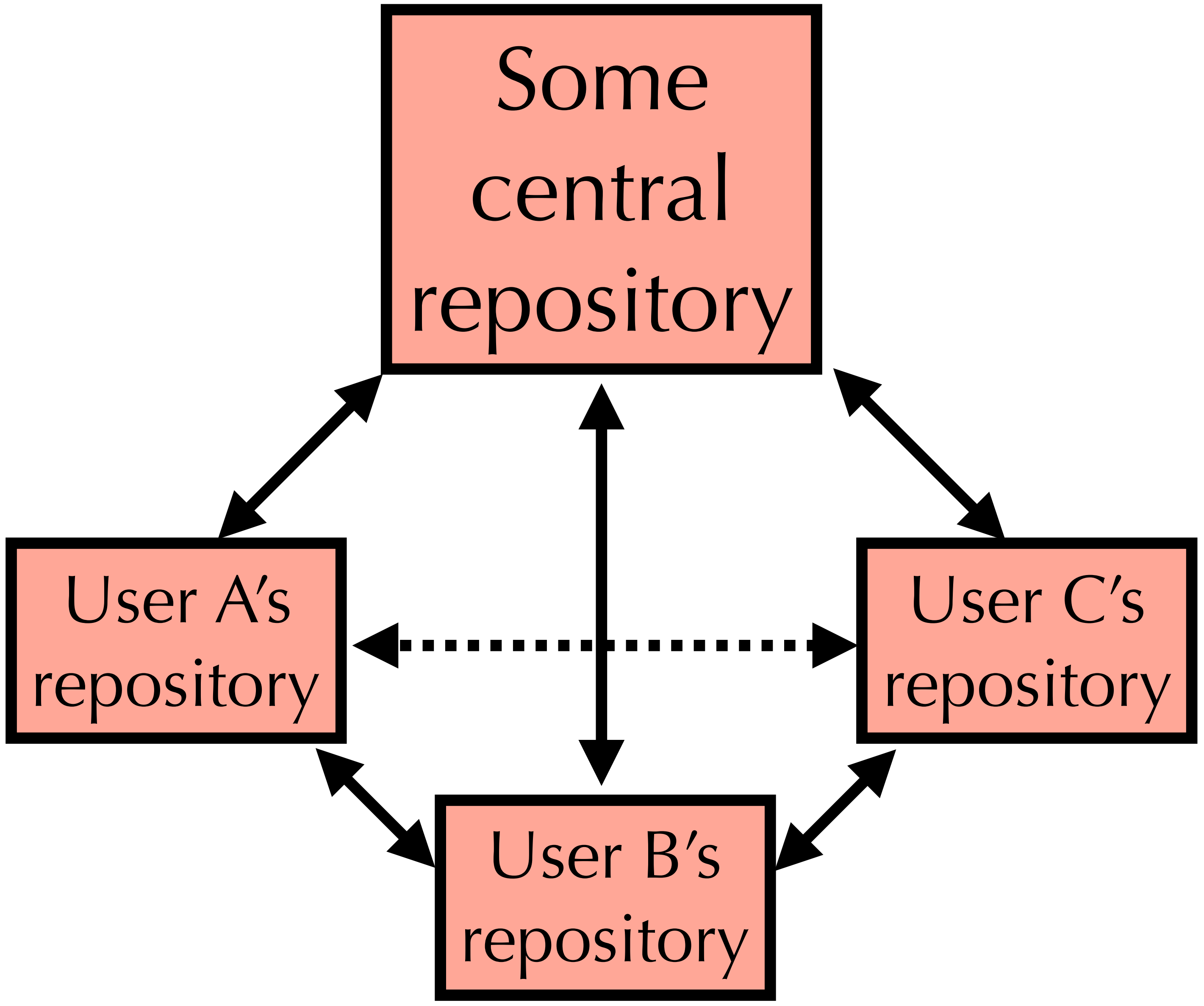
* Has anyone here used **Git** on their own projects? 😃

* Does anyone use a different **VCS** (Version Control Software) other than Git? (e.g. **SVN**, **CVS**, **Mercurial**) 😐

* Does anyone **NOT** use any VCS? 😱

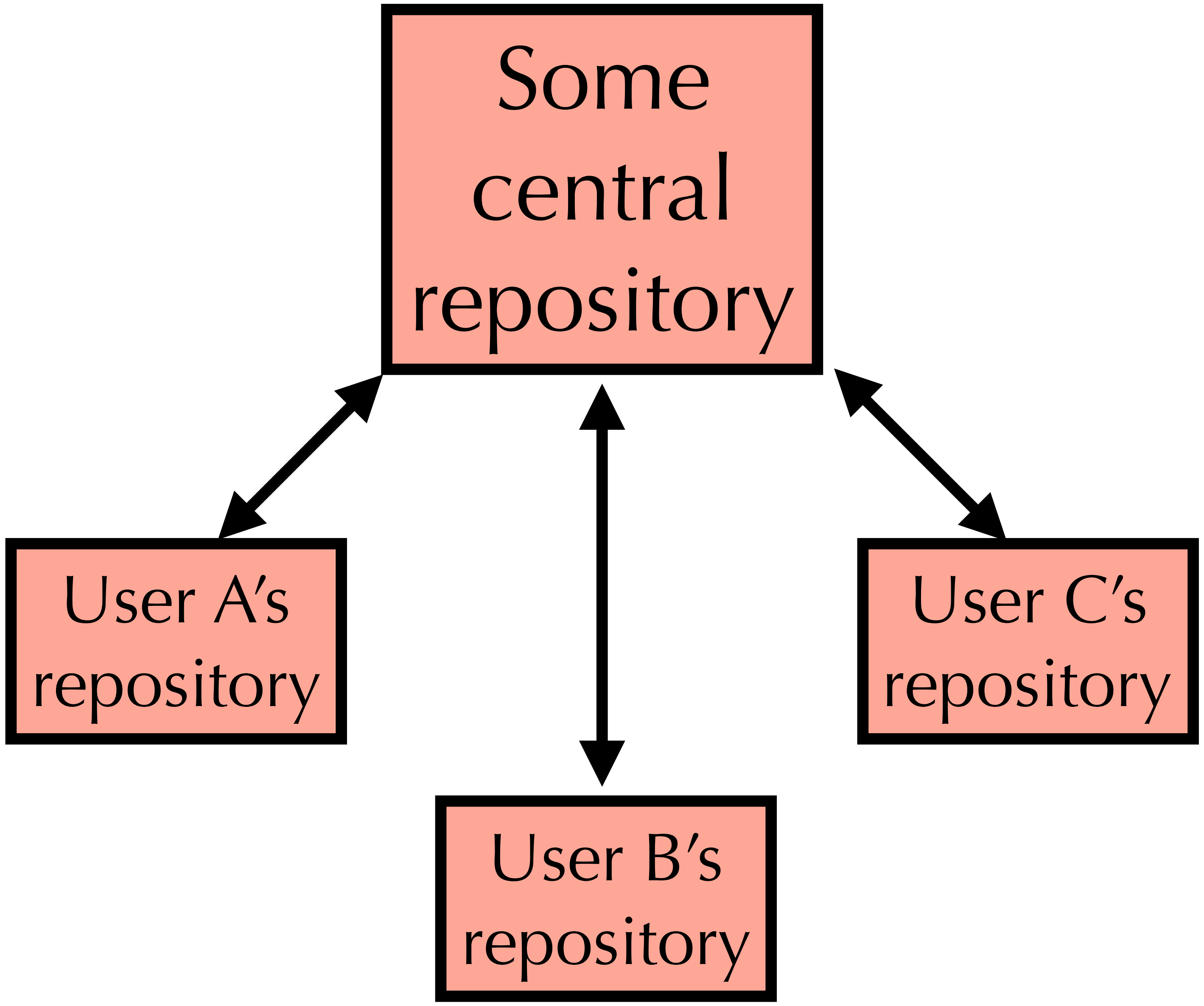
Centralised vs Distributed VCS



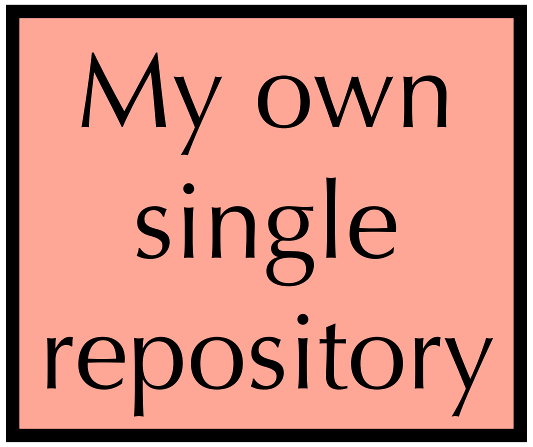


Most common ways of using Git

1. Effective central repository



2. Stand-alone



Git GUIs

* If you don't like the command-line and prefer to use a GUI, there are a large number of Git GUIs available
    * gitk
    * GitKraken
    * Atlassian SourceTree (Windows and Mac only)
    * Plus many more (but NOT necessaily for free)
* Some aspects (e.g. branches, merging, stashing) can be easier to manage using GUIs

Git config options

* Username

```
git config --global user.name “John Doe"
```

* E-mail address

```
git config --global user.email johndoe@example.com
```

* Default editor

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

* To check all options

```
git config --list
```

Git primer : Adding files and commiting

- Convert a regular directory into a git repository

```
git init
```

- Add files to the git repository

```
git add file1 file2 file3 …
```

```
git add src/subdirectory
```

```
git add files.????? files.*
```

- Finally commit all files to the repository

```
git commit -m “My very first git commit“
```

Every commit gets a 40-character long hash id, e.g. 6b6edb08dd4316d1ee682f3ff3a94b4205e02f75

Git primer : Diagnostic commands

* To create a (long) list of all the commits of the repository (most recent commits first)

```
git log
```

```
git log —stat
```

More info for each commit

* To get a summary of the current status of the repository

```
git status
```

* To view the difference between recently changed files and those in the repository

```
git diff
```

```
git diff filename
```

Git primer : Making and commiting changes to the local repository

* Make changes to and save any files.

* Inform git of files to be included in next commit (staging)

```
git add modifed_file1 modified_file2
```

* If you wish to move or remove any files

```
git mv filename newname
```

```
git rm filename
```

* Finally commit all files to the repository

```
git commit -m “My first git changes“
```

* Note : shortcut to add and commit all modified files in one go

```
git commit -am “Adding and commiting my first changes“
```

The '.gitignore' file

* 'git status' tells you the status of every file in the git directory, even when you don't want to know about it
    * e.g. Intermediate object files (when compiling), backup files, output generated by the code, other unneeded files
* To tell git to ignore certain filetypes, create a file in the main git directory called '.gitignore'
* Add the names of any files you wish to ignore
* Now everything should be a bit cleaner
* p.s. remember to actually add and commit the .gitignore file to the repository

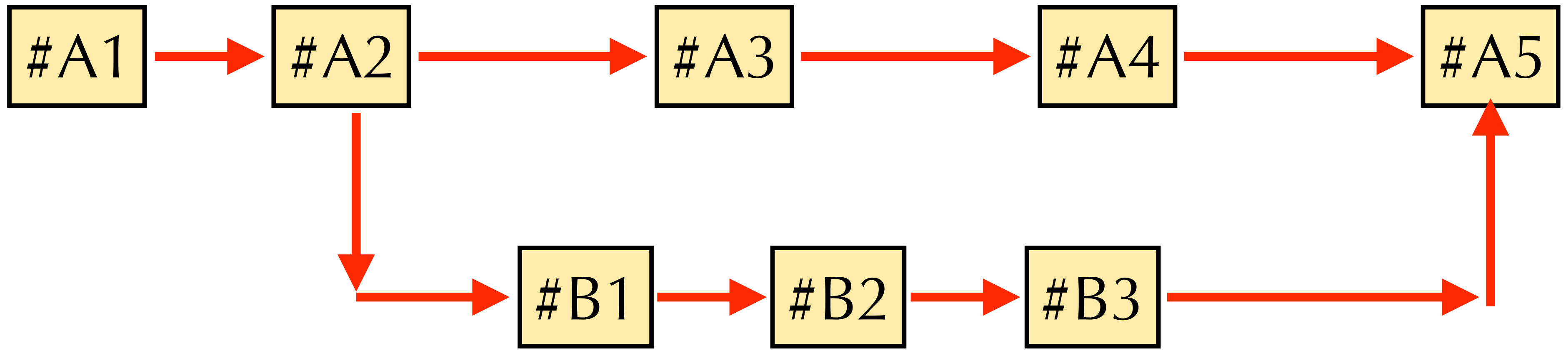
Branches

* Your Git repository can be split into different branches and then merged back together at a later time
* Why do this??
    * Work on a development feature over some timescale in parallel (merge once it's ready)
    * Need to add/test some bug fixes quickly
    * Want to experiment with some new brilliant idea but don't want to pollute the main branch (can delete branch if it's actually rubbish)

Branches

* The default/main branch is called the master branch

Master :



* With multiple branches, the HEAD is a pointer to the currenty active branch and commit

Branches

* To list all local branches (asterix next to currently active branch)

```
git branch
```

* To create a new branch (from current branch) and change to it

```
git branch devel
git checkout devel
```

```
git checkout -b devel
```

Combine both steps

* To delete a branch at any point

```
git branch -d oldbranch
```

Merging branches

- Merging one branch to another means to apply all unique commits in the second branch back to the first branch
- If all the commits affect different parts of the files, then this is trivially done. First change to the target branch (i.e. the one we wish to merge into)

```
git checkout targetbranch
```

- Then merge in all commits from the source branch

```
git merge sourcebranch
```

```
git rebase sourcebranch
```

Different type of merge

When merging goes wrong : Conflicts!

* If you change the same line in different commits and then try to merge, chaos ensues!

How to resolve conflicts

* Git will mark out in the file the conflicted lines with HEAD and the commit hash id (and a string of the log)

* To resolve the conflict, we must simply open the file and choose which of the two options we wish to retain

* Next, save and close the file, then add and commit to tell git the conflict is officially resolved

* In GUIs/IDEs, sometimes there are simple ways to select the chosen lines and commit the corrected files

Github

* Github is a code hosting web service, using Git (obviously) as the VCS
* Free for publically available projects
* Private repositories cost 7+ Dollars (approx 5.70+ Euros) per month (unlimited)

Setting up a Github repository

* Get a Github account
* Go to 'Your Profile' -> 'Repositories'
* Click 'New'
* Give your repository a name and select all basic settings (e.g. public or private)
* Clone your (empty) repository to your local machine

```
git clone https://github.com/username/reponame.git
```

Remotes

* To be able to access external repositories (either on your local machine, or on a server), you need to set-up a remote repository

* When you use 'git clone', the remote is automatically set-up for you

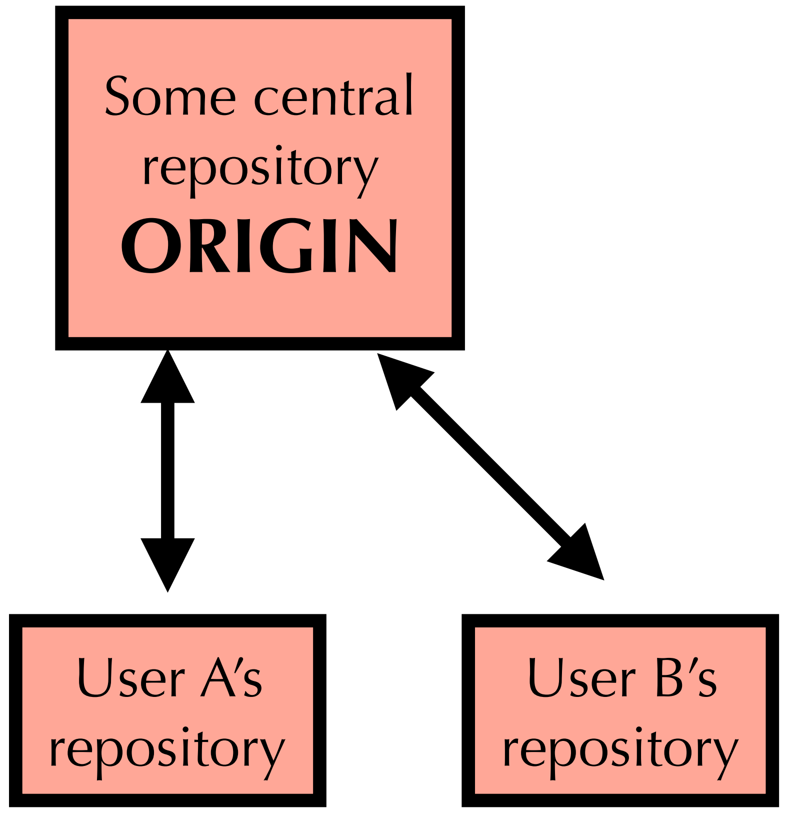
* By default, the 'central' repository is called 'origin'

* To see your remote repositories

```
git remote
```

```
git branch -vv
```

Gives info on remotes for branches



Working with remotes

* To retrieve an updated version of the repository from the remote

```
git fetch
```

* To update a specific repository from the remote repository

```
git pull origin reponame
```

Note : This is like a git merge
Can lead to conflicts!!

* To update the remote repository with your own local commits

```
git pull origin reponame
```

Must first update before pushing

```
git push origin reponame
```

Pull requests

* Often only certain users will have admin rights to push to the main repositories
* Regular users can submit a 'pull request' to the github repository
* This is like sending code to be included but it must be 'approved' by the main admin users before it can be included
* p.s. Regular users might need to fork the project rather than just a regular 'git clone'

Undoing things with Git

* Undoing things with Git can lead to a world of pain so be careful!

* If you've changed a single file but have changed your mind and want to go back to the original version

```
git checkout — filename
```

* If you've already added a file with 'git add' but have changed your mind and wish to undo that

```
git reset HEAD filename
```

Undoing things with Git

- If you've editted many files and need to update (via git pull) but don't want to commit yet

```
git stash
```

Puts all changes into a local patch file

- To re-add the changed code afterwards

```
git stash pop
```

```
git stash apply
```

Same as pop except it does not delete stash file

Git tips

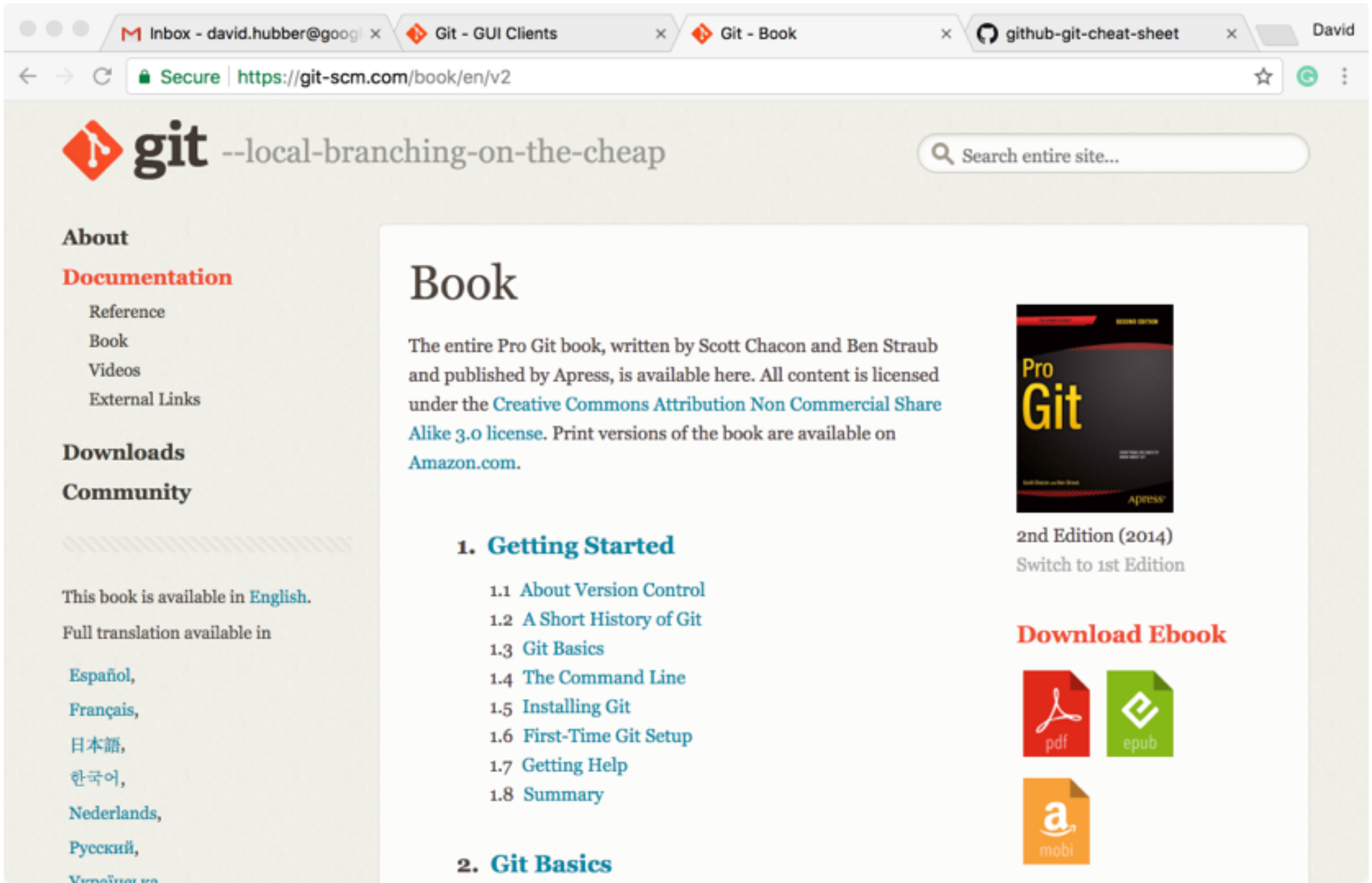
- Make a single commit per new or modified feature
  - If you decide to remove this feature, or find that you've introduced a bug, it's much easier to identify it and purge it from existence if it's contained in a single commit
  - Easier to write concise accurate logs with one or few features per commit
- Always make a new branch before adding a new major development feature, espeically if its experimental
  - If you find its a disaster, you can simply delete the branch and not have your git history contaminated by dead, useless code

Git tips

- Do NOT commit binaries and other large data files (unless absolutely necessary)
    - Git records a history of all added files and the Git metadata can become very big indeed if many large files (which might change with every commit) are added
- Use git stash for emergencies (e.g. when merging, pulling/updating) but try to reapply the stash asap
- Don't try and leave it too long between merges. Always try and update the development branch whenever the master is itself updated (e.g. by a merge with another branch)

Git online book

https://git-scm.com/book/en/v2



This book is available in English.

Full translation available in

Español,
Français,
日本語,
한국어,
Nederlands,
Русский,

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the Creative Commons Attribution Non Commercial Share Alike 3.0 license. Print versions of the book are available on Amazon.com.

2nd Edition (2014)

Switch to 1st Edition

Download Ebook