Cardiff's School of Computer Science & Informatics

CM1102 "Web Applications"

Version Control with git

Lab 1

Dr Natasha Edwards

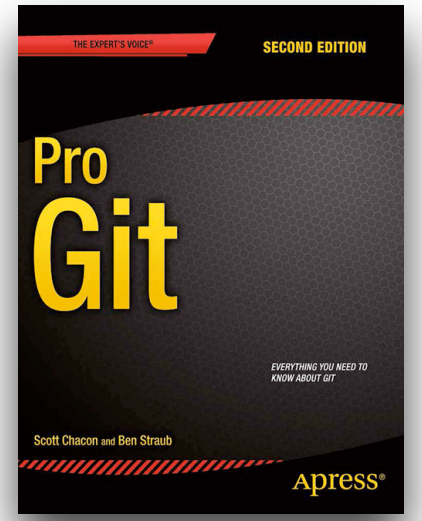
Learning objectives

- Learn and practice how to use git for version control, working with local and remote repositories

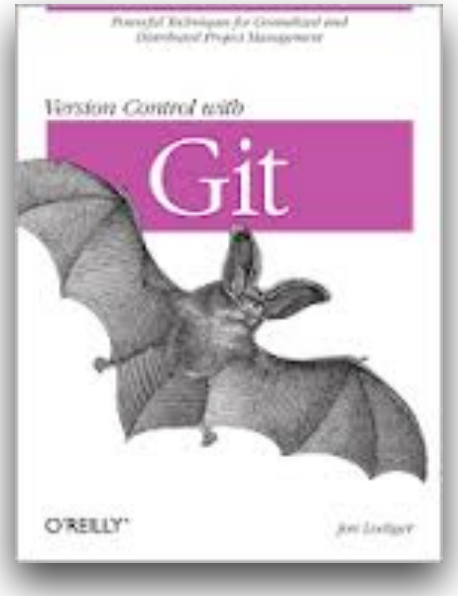
Resources

- **'Pro Git' book**
  - 2nd ed, 2014, available at:
    - https://git-scm.com/book/en/v2
  - of particular relevance Ch. 2 and parts of 5

    (but other chapters might be of interest if you want to learn more)



- Other (printed) books, e.g.



Working smarter, not harder!

- Using Command line
  - e.g. Windows, see: 'MS-DOS and command line overview' -> http://www.computerhope.com/overview.htm
- Documentation on our systems, software, etc: http://docs.cs.cf.ac.uk/
- COMSC's GitLab
  - https://gitlab.cs.cf.ac.uk/users/sign_in
  - Basic intro: http://docs.cs.cf.ac.uk/notes/using-gitlab/

Basic Git Commands

- Create your repository (repo)
- Copy (clone) your repo
- Add a file to your repo
- Configure the commit author
- Make a commit
- View your commits
- View commit differences (diff)
- Remove and rename files

More "Advanced" Git

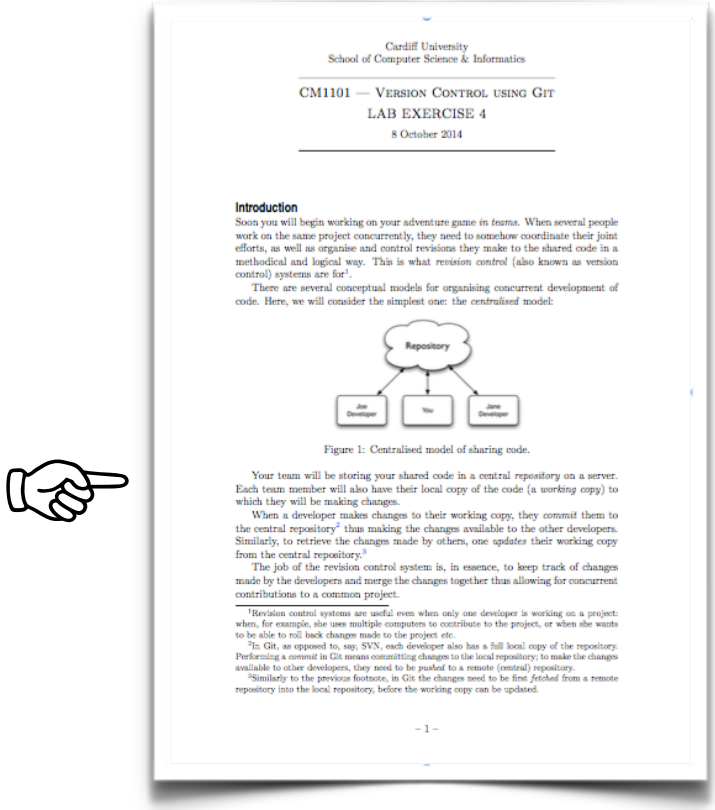
- Branch
- Merge/ Rebase
- Remote Git
- Distributed Git

In this module, we will consider **`remote git`**, but not the other topics (these will be explored in other modules).

The Git Command Line

Some commands should be already familiar to you…

"Computational Thinking" module/ "Git Lab" in Week 4



Cardiff University
School of Computer Science & Informatics

CM1101 — Version Control using Git
LAB EXERCISE 4
8 October 2014

Introduction

Soon you will begin working on your adventure game *in teams*. When several people work on the same project concurrently, they need to somehow coordinate their joint efforts, as well as organise and control revisions they make to the shared code in a methodical and logical way. This is what *revision control* (also known as version control) systems are for[1].

There are several conceptual models for organising concurrent development of code. Here, we will consider the simplest one: the *centralised* model:

Figure 1: Centralised model of sharing code.

Your team will be storing your shared code in a central *repository* on a server. Each team member will also have their local copy of the code (a *working copy*) to which they will be making changes.

When a developer makes changes to their working copy, they *commit* them to the central repository[2] thus making the changes available to the other developers. Similarly, to retrieve the changes made by others, one *updates* their working copy from the central repository.[3]

The job of the revision control system is, in essence, to keep track of changes made by the developers and merge the changes together thus allowing for concurrent contributions to a common project.

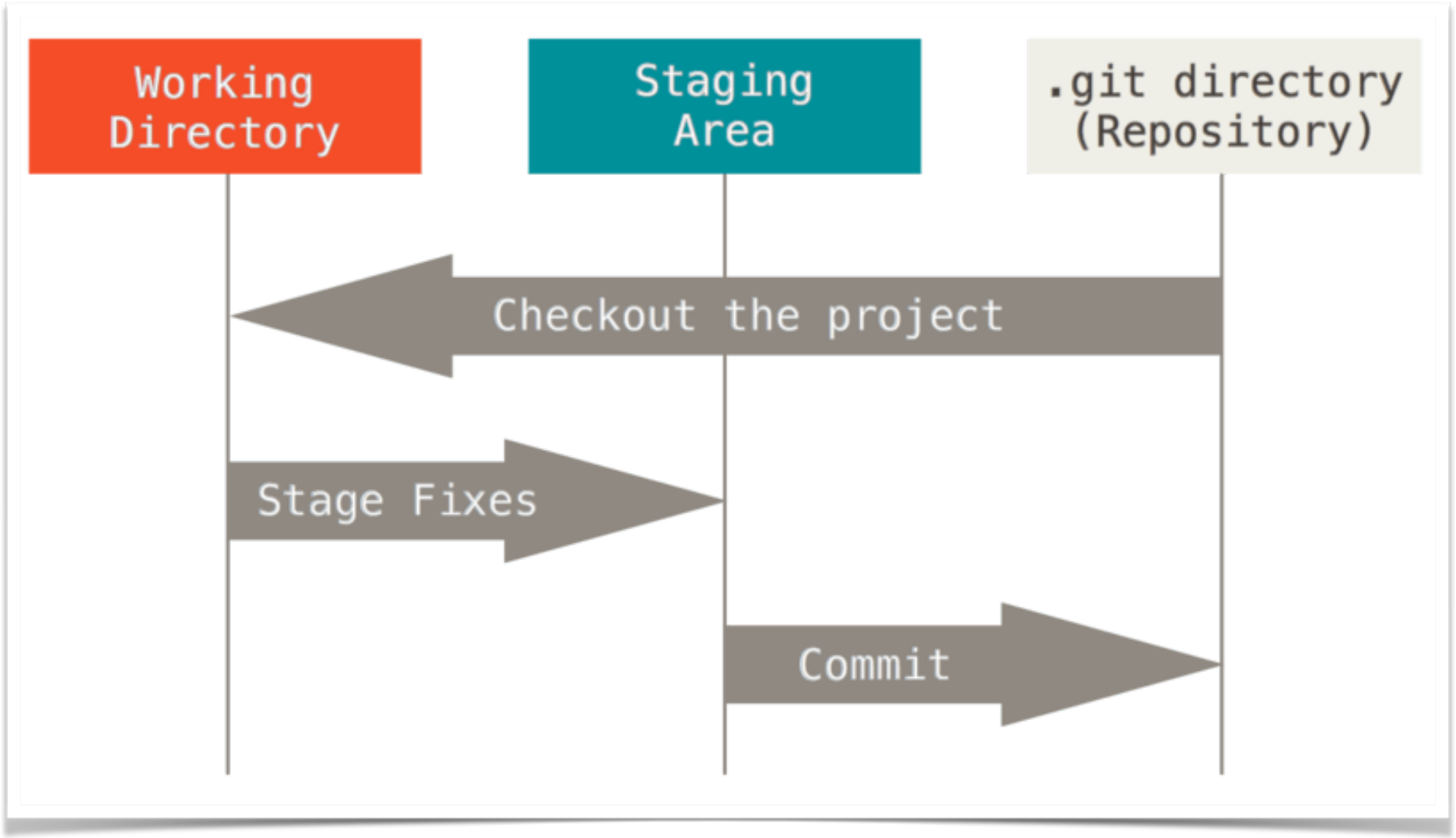
[1]Revision control systems are useful even when only one developer is working on a project: when, for example, she uses multiple computers to contribute to the project, or when she wants to be able to roll back changes made to the project etc.

[2]In Git, as opposed to, say, SVN, each developer also has a full local copy of the repository. Performing a *commit* in Git means committing changes to the local repository; to make the changes available to other developers, they need to be *pushed* to a remote (central) repository.

[3]Similarly to the previous footnote, in Git the changes need to be first *fetched* from a remote repository into the local repository, before the working copy can be updated.

– 1 –

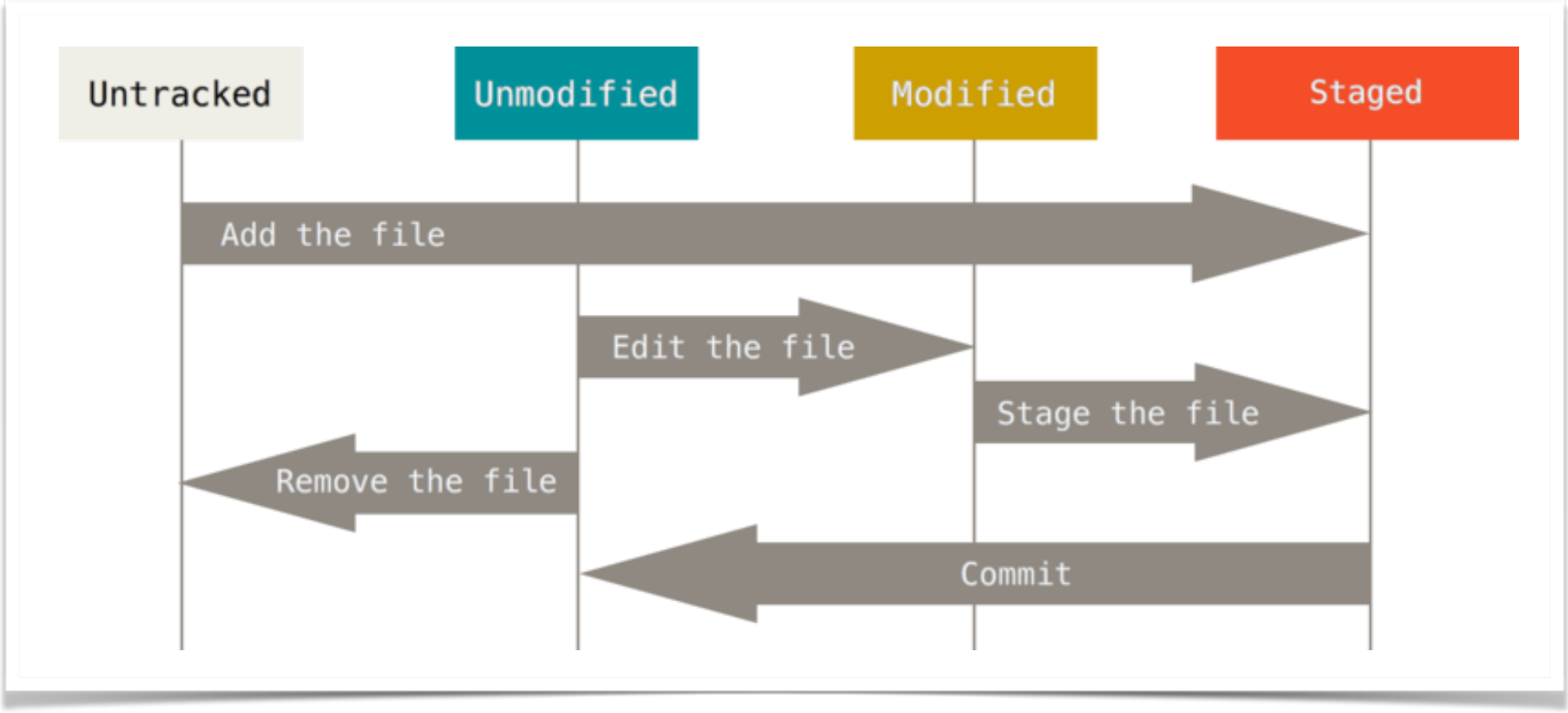
Main Sections of Git Project



Three States of Git

- Modified
  - you made changes to a file, but not committed to the DB yet
- Staged
  - you have marked a modified file in its current version to go into your next commit snapshot
- Committed
  - your data is safely stored in DB

Lifecycle of the status of your files



Working with a Repo

to start tracking a project:

```
$ git init
```

to start tracking files:

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

to clone a repo:

```
$ git clone https://github.com/libgit2/libgit2
```

you can clone a repo to a specific directory of your choice:

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

Recording Changes

to check the status of your files (e.g. *staged, committed)*:

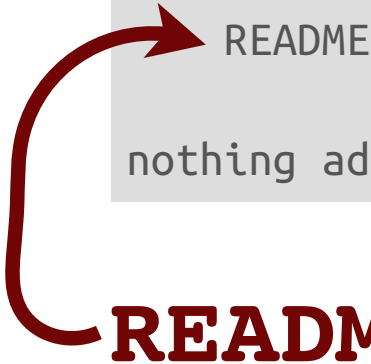
```
$ git status
On branch master
nothing to commit, working directory clean
```

no modified or tracked files

```
$ echo 'My Project' > README
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```



README
is untracked

Tracking New Files

```
$ git add README
```

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

README is tracked and staged, and now needs to be committed

Staging Modified Files (1)

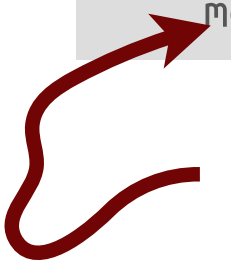
Say, we modified the already tracked **`CONTRIBUTING.md`** file. When you run **`git status`** command, the output look like:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

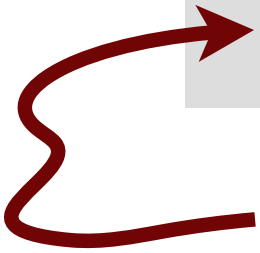


Tracked file `Contributing.md` has been modified but NOT staged

Staging Modified Files (2)

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```



Both files are now staged

Staging Modified Files (3)

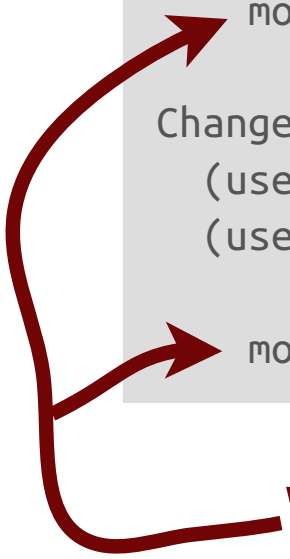
Say, we forgot to change something in **`CONTRIBUTING.md`** file and modify it again. Let's run **`git status`** command again:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```



What is going on?

If you run `git commit`, which one will be committed?

Staging Modified Files (4)

You need to run **`git add`** first:

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```

Short Status

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

Left column: file is staged; the right column: file is modified

A - Added to the staging area

M - Modified

MM - Modified, staged, modified (contains changes that are both, staged and unstaged)

?? - non-tracked files

Ignoring Files

- You may want to ignore some files, e.g. log files, system and temporary files, etc.
- For this purpose, create a **`.gitignore`** file:

```
$ cat .gitignore
*.[oa]
*~
```

- Where do I put this file?
  - You can put a **`.gitignore`** in any (and every) single directory of your project.
  - However, better practice is to have one global .gitignore the project root directory
  - Also, possible to create one global local **`.gitignore`** file, that will manage all of your git repositories:

```
$ git config --global core.excludesfile ~/.gitignore_global
```

`.gitignore` file also need to be tracked and committed

Viewing Your Staged and Unstaged Changes (1)

- **`git diff`** command is used to see exactly what you have changed
- E.g. you edited and staged the **`README`** file again, and then edited the **`CONTRIBUTING.md`** file without staging it:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

Viewing Your Staged and Unstaged Changes (2)

- To see what you have changed but not yet staged, use command **`git diff`** command, which compares the data in your working directory with your staging area

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 8ebb991..643e24f 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -65,7 +65,8 @@ branch directly, things can get messy.
 Please include a nice description of your changes when you submit your PR;
 if we have to read the whole diff to figure out why you're contributing
 in the first place, you're less likely to get feedback and have your change
-merged in.
+merged in. Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.

 If you are starting to work on a particular area, feel free to submit a
 that highlights your work in progress (and note in the PR title that it
```

`git diff` doesn't show all the changes since your last commit - only changes that are still unstaged

Comparing Your Staged Changes With Your Last Commit

- To see what you have staged that will go into your next commit: use **`git diff --staged`** command:

```
$ git diff --staged
diff --git a/README b/README
new file mode 100644
index 0000000..03902a1
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+My Project
```

`git diff` doesn't show all the changes since your last commit - only changes that are still unstaged

Viewing Your Staged and Unstaged Changes

- Another example: stage the **`CONTRIBUTING.md`** file and then edit it, then use **`git diff`** to see the staged and unstaged changes:

```
$ git add CONTRIBUTING.md
$ echo '# test line' >> CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

Viewing Your Staged and Unstaged Changes

- Use **`git diff`** to see what is still unstaged:

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 643e24f..87f08c8 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -119,3 +119,4 @@ at the
 ## Starter Projects

 See our [projects list](https://github.com/libgit2/libgit2/blob/development/PROJECTS.md).
+# test line
```

Viewing Changes Your Staged So Far

- Use **`git diff --cached`** to see what you have staged so far:

```
$ git diff --cached
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 8ebb991..643e24f 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -65,7 +65,8 @@ branch directly, things can get messy.
 Please include a nice description of your changes when you submit your PR;
 if we have to read the whole diff to figure out why you're contributing
 in the first place, you're less likely to get feedback and have your change
-merged in.
+merged in. Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.
```

Committing Changes

```
$ git commit
```

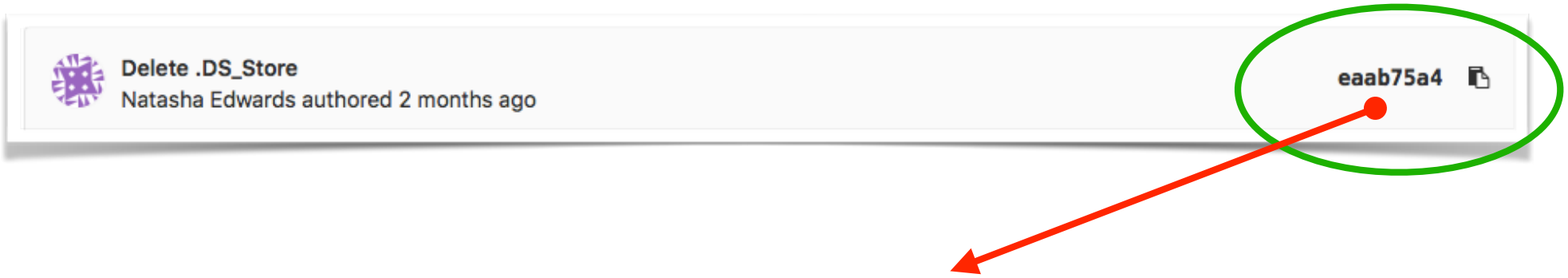
- Use a text editor of your choice to preview what you are committing, add notes/ messages, etc.
- Alternatively, you can do this '*inline*':

```
$ 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
```

- The output gives you: ***branch*** committed to, *SHA-1* ***checksum*** of the commit, ***number of files*** changed, ***statistics*** about lines added and removed in the commit.

Checksum

- A small-sized datum that is used to detect errors, which may have occurred during the transmission of the data digitally.
- several algorithms
- git's checksum: SHA-1 hush
  - checksum of the content and its header



eaab75a4d812e249098a62ef102791a186ecf1e2

Committing Changes: Skip the Staging

Already tracked files can be committed without being staged:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

     modified:   CONTRIBUTING.md

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
 1 file changed, 5 insertions(+), 0 deletions(-)
```

Meaning: no need to run **`git add`** on the "**`CONTRIBUTING.md`**" file in this case before you commit.

Removing Files from Git

- To remove a file from Git, you need to remove it from the staging area by using **`git rm`** and then commit

If you run [Unix] **`rm`** <file> command, the **`PROJECTS.md`** file will be removed from your working directory, but will show as "unstaged" changes.

Removing Files from Git

- To remove a file from Git, you need to remove it from the staging area and then commit.
  - removing it from the working directory

```
$ 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 <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    PROJECTS.md

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

Removing Files from Git

- Complete removal from Git:

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

    deleted:    PROJECTS.md
```

`PROJECTS.md` file will be deleted

- Partial removal: keep the file in the working directory, but tell Git not to track it:

```
$ git rm --cached README
```

- You can pass files, directories patterns

Moving (or rather renaming!) files

- Git doesn't explicitly track file movement. Renaming a file in Git does not store the metadata that tells Git you renamed a file.
- Confusingly, **`git mv`** command is more about renaming rather than moving files

```
$ git mv README.md README
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

UNDO

- Perhaps, the most important command? **BUT** needs extra care as some undos can not be undone!
- To try the previous commit, e.g. you commit too early and forgot to add files:

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

e.g.

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

this requires only one, single commit!

- To unstage a staged file:

```
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M       CONTRIBUTING.md
```

This changes the file to *"modified"* but *"unstaged"*

UNDO

- To unmodify a modified file:

```
$ git checkout -- CONTRIBUTING.md
```

Warning! This is a dangerous command! You loose the changes as the file is overwritten! Might be better off using branching (*NB: not covered in this module*).

Commit History

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

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

Quite a few output options - see pp. 58 - 65 of the textbook

Tags

Tag points to a specific commit in the repo.

```
$ git tag -a v1.4 -m 'my version 1.4'
$ git tag
v0.1
v1.3
v1.4
```

tagging message, stored with the tag

To see the tag data along with the commit:

```
$ git show v1.4
tag v1.4
Tagger: Ben Straub <ben@straub.cc>
Date:   Sat May 3 20:19:12 2014 -0700

my version 1.4

commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number
```

You can also have lightweight tags; tag later or share tags - see pp. 75 - 78 of the textbook

Aliases

Productivity!

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

E.g., own "unstage" alias:

```
$ git config --global alias.unstage 'reset HEAD --'
```

```
$ git unstage fileA
$ git reset HEAD -- fileA
```

equivalent commands

Remote git: remote repos

Clone a remote repo

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd ticgit
$ git remote
origin
```

To see the URLs' shortnames:

```
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
```

this command will show all your remotes

Adding Remote Repos and Fetching

Clone a remote repo:

```
$ git remote
origin
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)
```

shortname

To fetch the information:

```
$ git fetch pb
remote: Counting objects: 43, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 43 (delta 10), reused 31 (delta 5)
Unpacking objects: 100% (43/43), done.
From https://github.com/paulboone/ticgit
 * [new branch]      master     -> pb/master
 * [new branch]      ticgit     -> pb/ticgit
```

branches are now accessible locally

Pushing to Remote Repos

```
$ git push origin master
```

Only works if you cloned from a server to which you have write access and if nobody pushed in the meantime

Inspecting a Remote Repo

To get more information about a remote:

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master                               tracked
    dev-branch                           tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Renaming and Removing Remote Repos

To change a shortname:

```
$ git remote rename pb paul
$ git remote
origin
paul
```

this also changes your remote branch names

To remove a remote:

```
$ git remote rm paul
$ git remote
origin
```

Alternatives to git

- Subversion
- CVS
- Perforce
- Bazaar
- …
- http://alternativeto.net lists 26 (open source & commercial)

Way Forward

- Future work:
  - Advanced git: branching, merging, dealing with merge conflicts, rebasing, distributed git