CAB302 Software Development
Lecture 7 — Source Control

Faculty of Science and Engineering

Semester 1, 2016

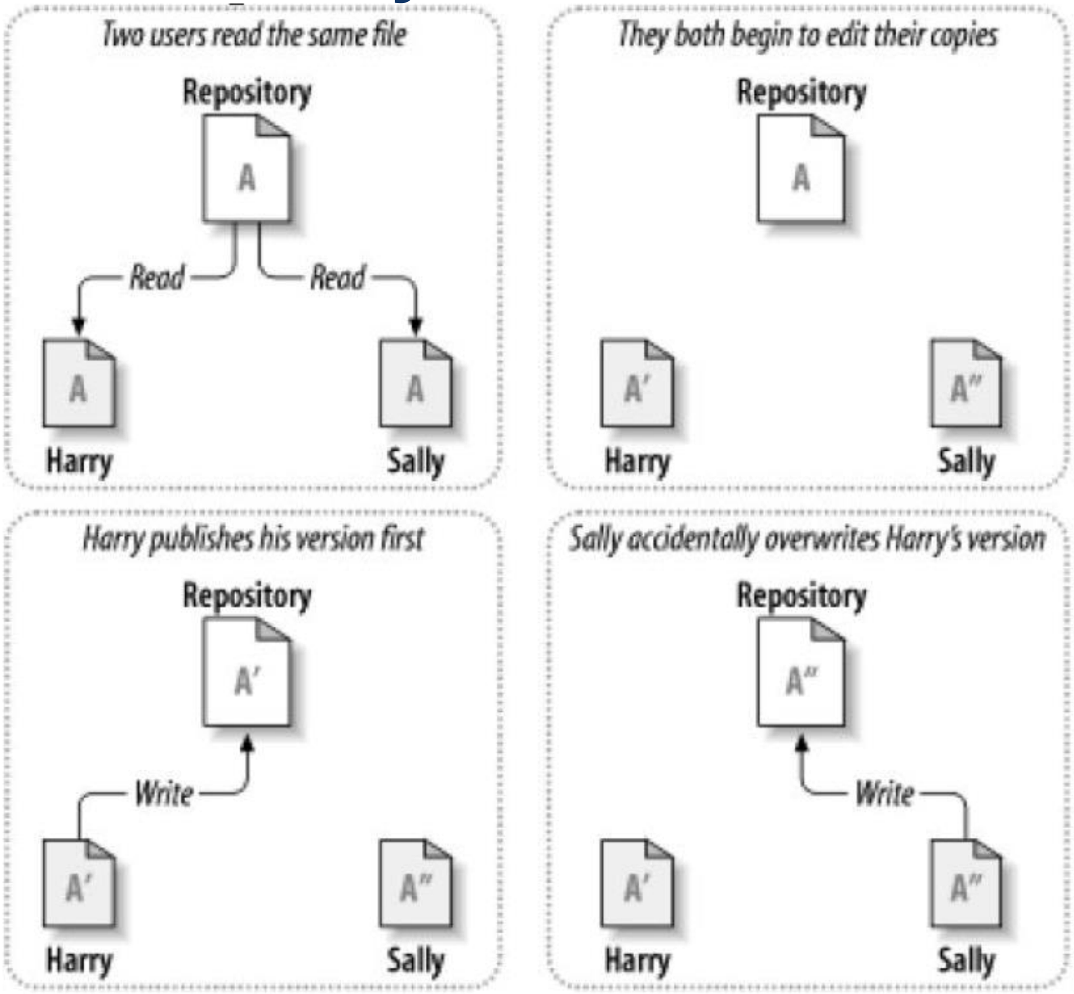
Aims of the Week 7 lecture and practical session

- To understand the history and different approaches to source control and version management
- To understand the most important concepts common to source control systems – especially commits, staging, file differences and patches, repositories and repository snapshots.
- To understand the consequences of poor version management and the conflicts that may arise
- To get practical experience of the use of an industry standard distributed source control system

Agenda

- Why source control matters
- Some history and some earlier approaches
  - The centralised diff and patch model
- Understanding the basics of Source Control via SVN
- The move to distributed source control
- Working with Git
- And working with Git

Why it Matters



Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

http://www.imdb.com/title/tt0098635/ Source: SVN Help

Why we need revision control

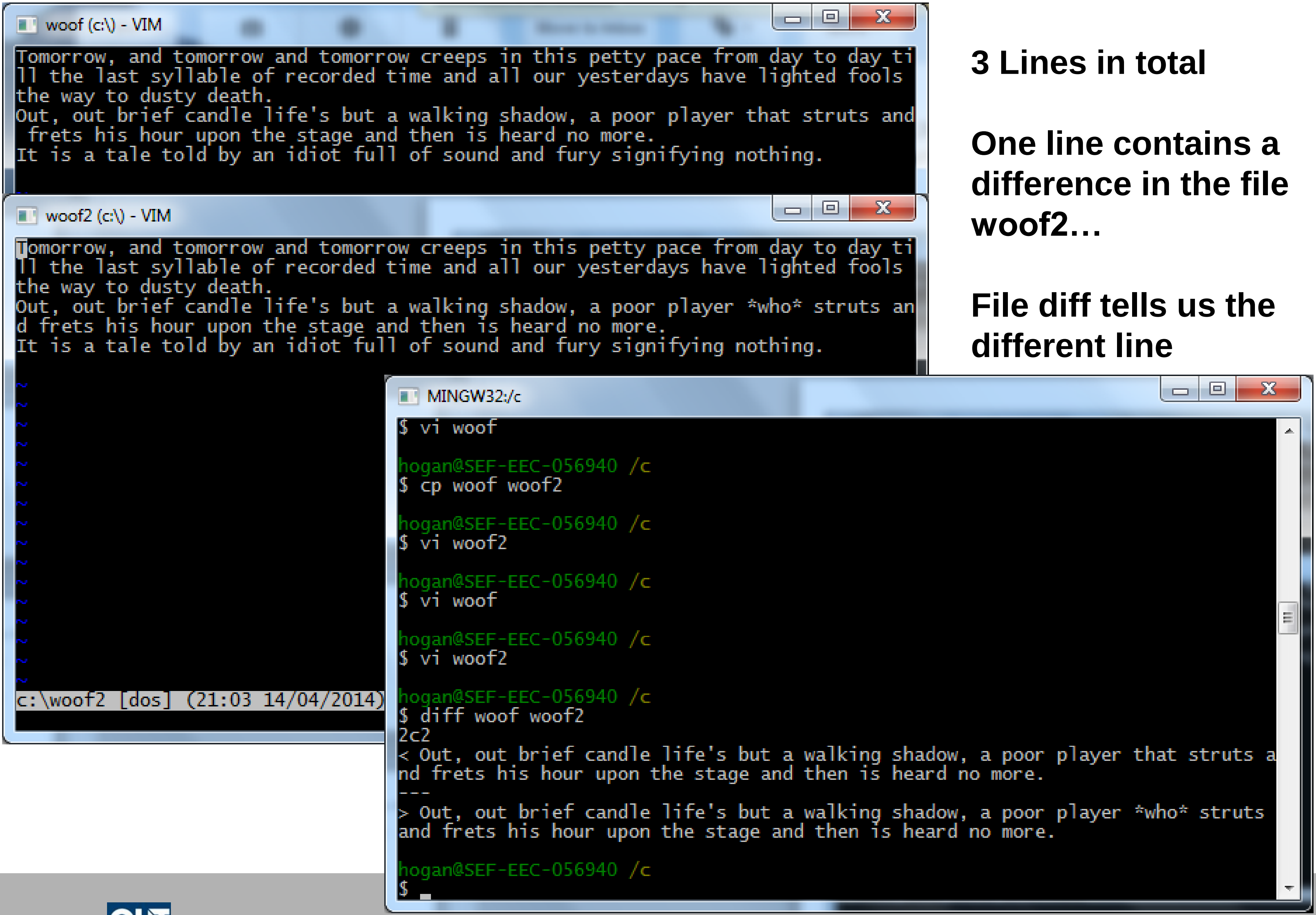
- Configuration management and source control are essential for any professional developer
- Even on small, single developer projects you should get into the habit of checking your code into a version control system
- It forces you to back up your work
- You have a *fortress of working code* to which you can retreat when the next changes make a mess of your system
  - This is incredibly important
  - Maintain both a development version and a stable version
- You can recreate any version of the system
  - Crucial for support of released products

Version control principles

- Source Control System cardinal rule #1:
  - It is okay to have a local source control system on your machine, but you really must have a repository or copy elsewhere
  - It doesn’t help much if your code is complete and beautifully versioned if your hard drive crashes
- Source Control System cardinal rule #2:
  - It is better to have too much stuff under version control than to have too little
  - Test source must be part of the version control story
  - Documentation, planning documents, etc, can all go into the mix as long as the structure is sensible

Some History

- Revision control systems arrived early, but still surprisingly late
- Quick history may be found here:
  - https://code.google.com/p/pysync/wiki/VCSHistory
  - We will play acronym guessing games in the lecture
  - Or you can look them up as you go
- We begin with SCCS (Bell Labs, 1972; IBM 370)
- RCS (early versions of unix, mid 1970s)
- Both of these offered revision control on a local system
- Maintain a careful database of revisions or patches
  - Keep the files, but also store the file diffs
  - Recreate any version at all by ‘adding up the diffs’



```
Tomorrow, and tomorrow and tomorrow creeps in this petty pace from day to day ti
ll the last syllable of recorded time and all our yesterdays have lighted fools
the way to dusty death.
Out, out brief candle life's but a walking shadow, a poor player that struts and
 frets his hour upon the stage and then is heard no more.
It is a tale told by an idiot full of sound and fury signifying nothing.
```

```
Tomorrow, and tomorrow and tomorrow creeps in this petty pace from day to day ti
ll the last syllable of recorded time and all our yesterdays have lighted fools
the way to dusty death.
Out, out brief candle life's but a walking shadow, a poor player *who* struts an
d frets his hour upon the stage and then is heard no more.
It is a tale told by an idiot full of sound and fury signifying nothing.

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
c:\woof2 [dos] (21:03 14/04/2014)
```

```
$ vi woof

hogan@SEF-EEC-056940 /c
$ cp woof woof2

hogan@SEF-EEC-056940 /c
$ vi woof2

hogan@SEF-EEC-056940 /c
$ vi woof

hogan@SEF-EEC-056940 /c
$ vi woof2

hogan@SEF-EEC-056940 /c
$ diff woof woof2
2c2
< Out, out brief candle life's but a walking shadow, a poor player that struts a
nd frets his hour upon the stage and then is heard no more.
---
> Out, out brief candle life's but a walking shadow, a poor player *who* struts
and frets his hour upon the stage and then is heard no more.

hogan@SEF-EEC-056940 /c
$ _
```

3 Lines in total

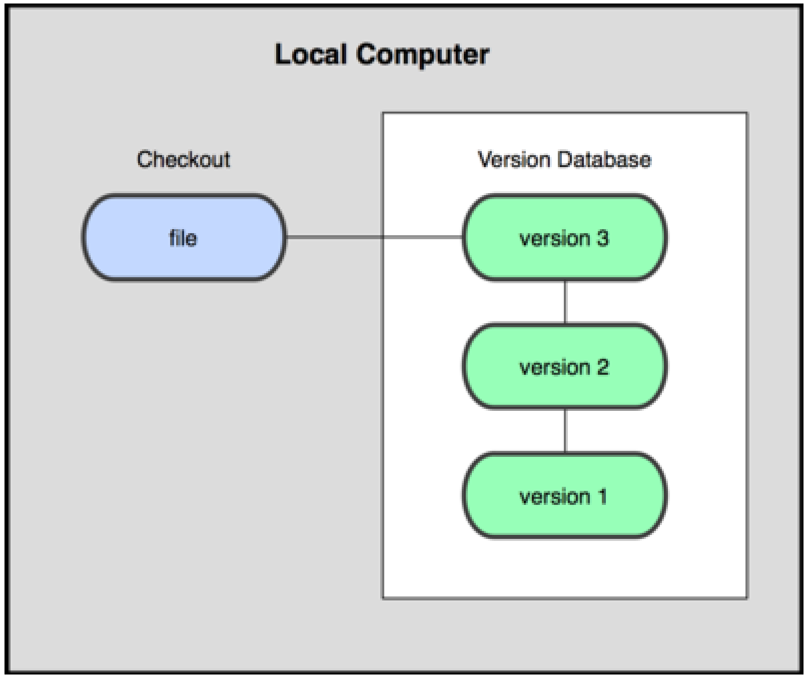
One line contains a difference in the file woof2…

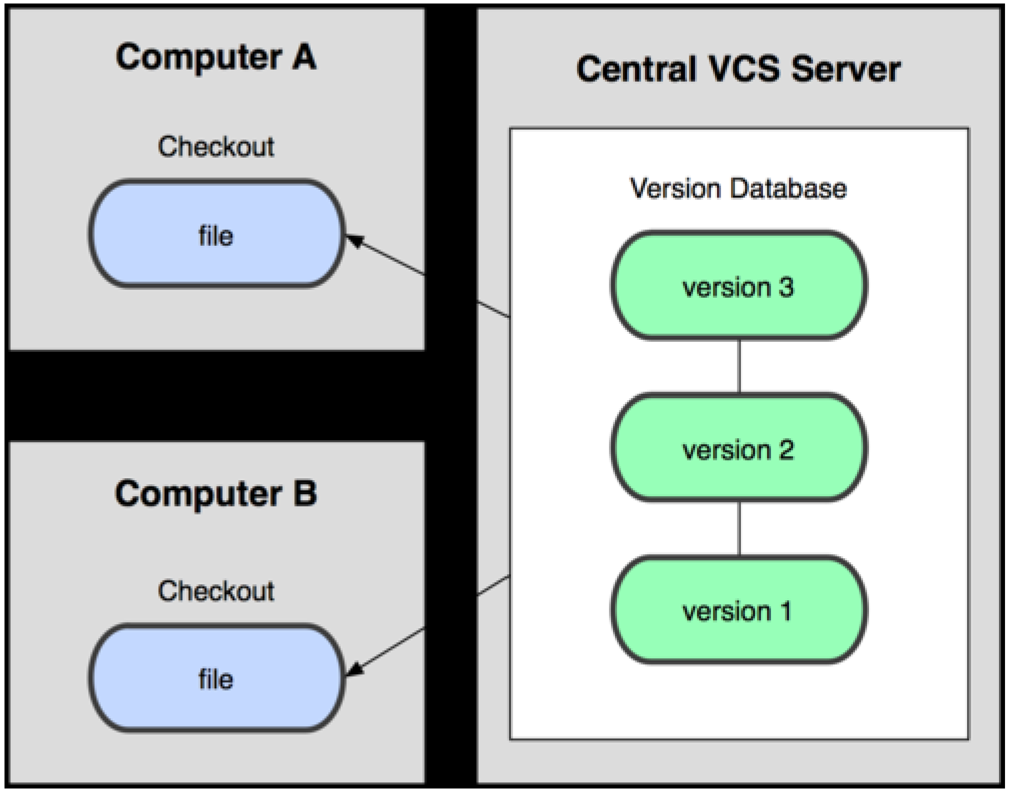
File diff tells us the different line

Some History (2)

- But locality has its limitations – we need to cater for a team
- CVS (successor to RCS, and a major system for many years)
  - First true central repo system in wide use
  - The first one we really taught in SE classes
  - Clearly legacy, but still in use through inertia
  - http://www.nongnu.org/cvs/
- Succeeded by SVN (Subversion) http://subversion.apache.org/
  - Still in very wide use (see a little later for details)
  - Great documentation: http://svnbook.red-bean.com/
  - Lots of tool support (Tortoise; plugins)
  - [Stack Overflow question was here, but now dead…]

Local vs Central Repositories



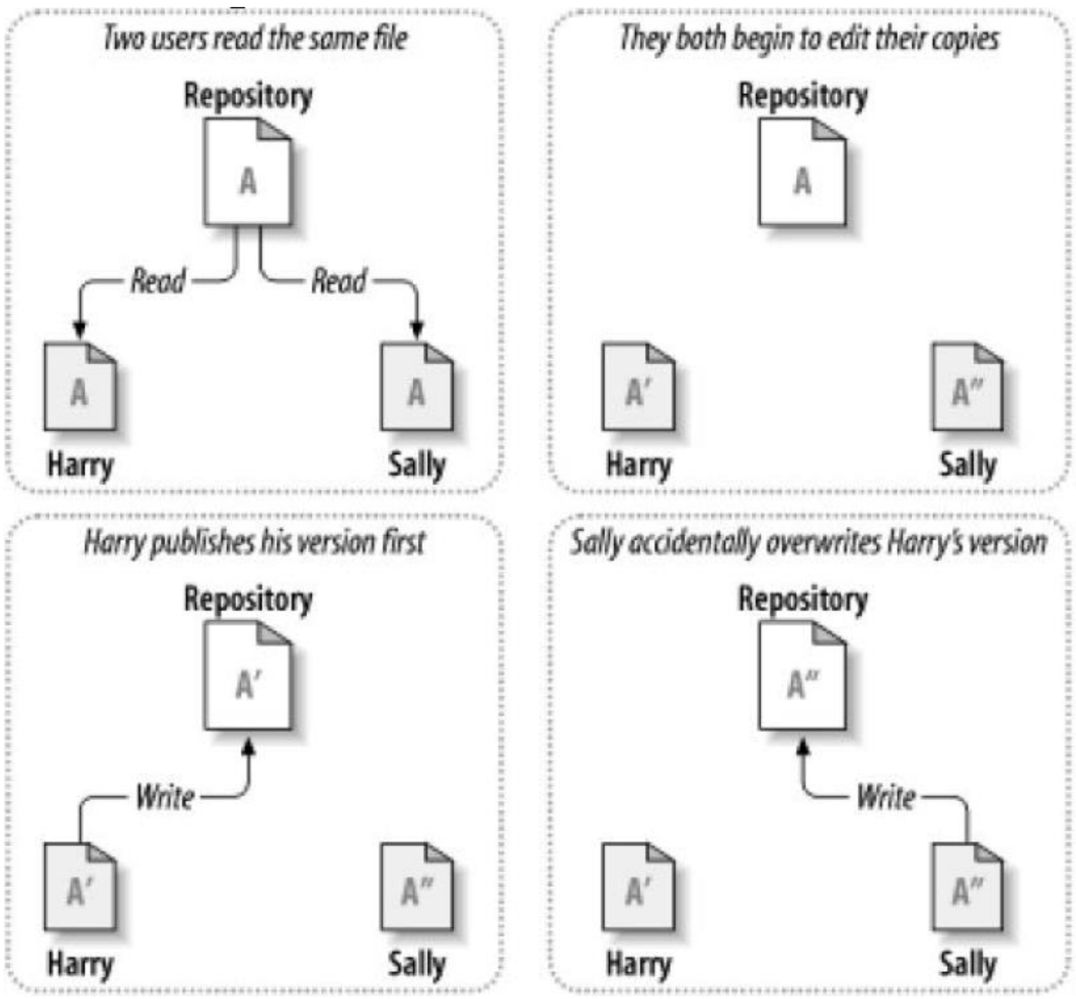


See chapter 1 of *The Git Book*: http://git-scm.com/book

The Story So far

- We have moved from local to central repositories
- But we have mainly kept faith with the patch model
- When we moved to the centralised model, we introduced the idea of a checkout – making a local copy of some code
- Hmmm, let’s think about this:
  - Any team member can check out *any* file from the repo
  - *So multiple* versions of the same file
  - Edited by *different* people?
  - At *different* times?
  - At the *same* time?

Version control principles

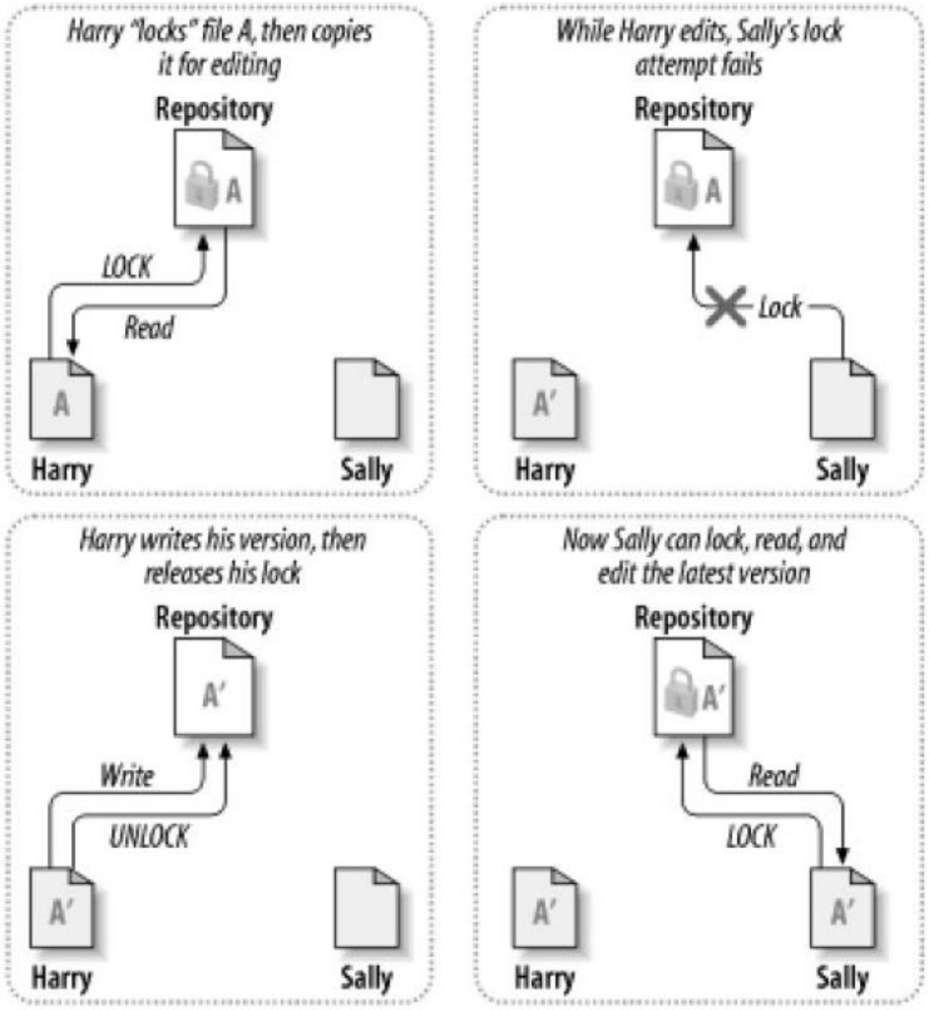

Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

Source: SVN Help

Version Control and Sanity

- This may not end well, even for Harry and Sally
- We can: Harmonise the changes somehow by making some sort of decision on which line wins and which one loses
  - [This involves a *merge of the code*]
  - Inevitably difficult to negotiate
- Or: Avoid the problem in the first place by ensuring that only one person can deal with each file at a time
  - [This involves the use of a *lock]*
  - *Needs to have software support*

Solution 2: The Lock


Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A′
Harry
Sally
Harry writes his version, then releases his lock
Repository
A′
Write
UNLOCK
A′
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A′
Read
LOCK
A′
A′
Harry
Sally

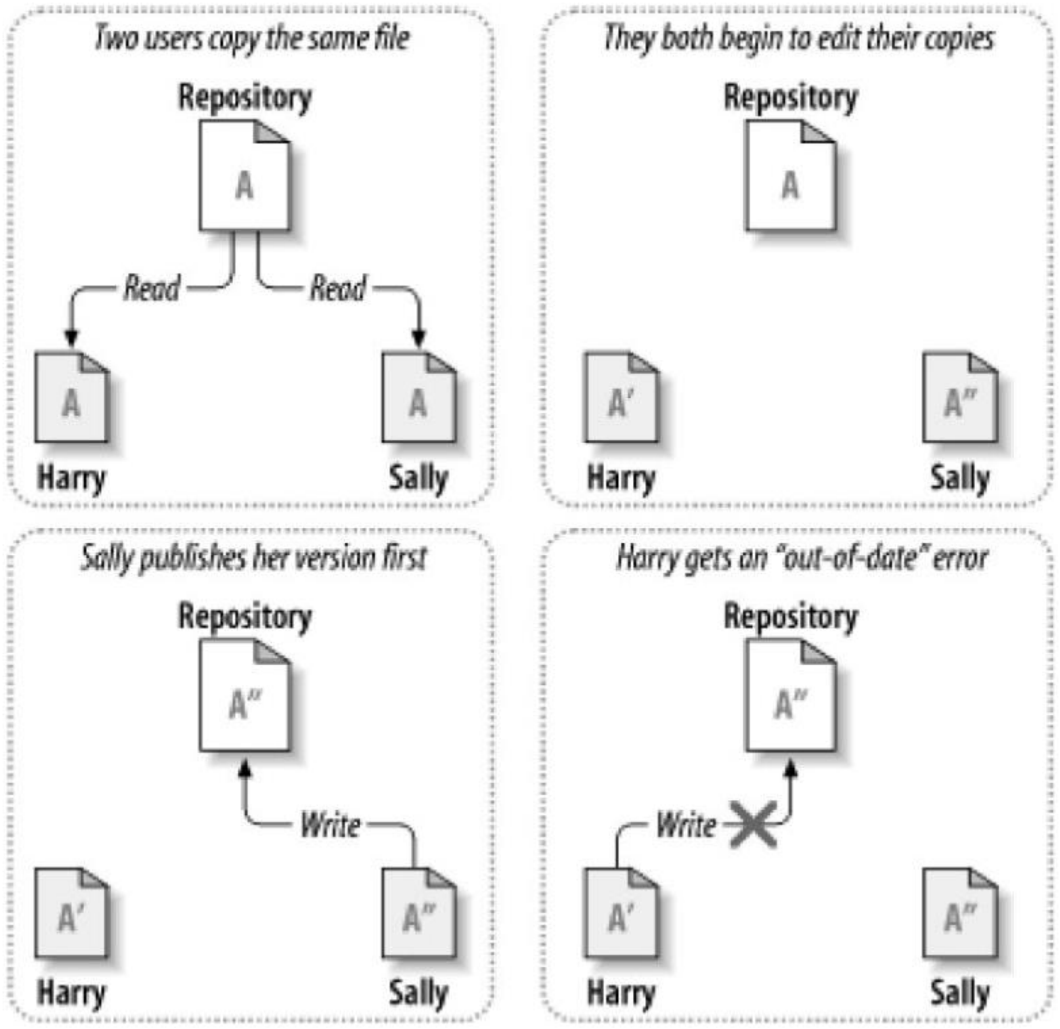
Source: SVN Help

The Lock

- In Summary: lock-modify-unlock
  - Ensure that only one developer is allowed to modify a particular class at a given time
  - Disadvantages:
    - Unnecessary serialisation of development steps
    - Programmers who forget to release their lock!
    - Doesn’t consider dependencies between classes that may both be out to different people at the same time

- [Note for later: this lock is directly analogous to the lock mechanism in threads – here the developer has the lock]

Solution 1: The Merge (1)

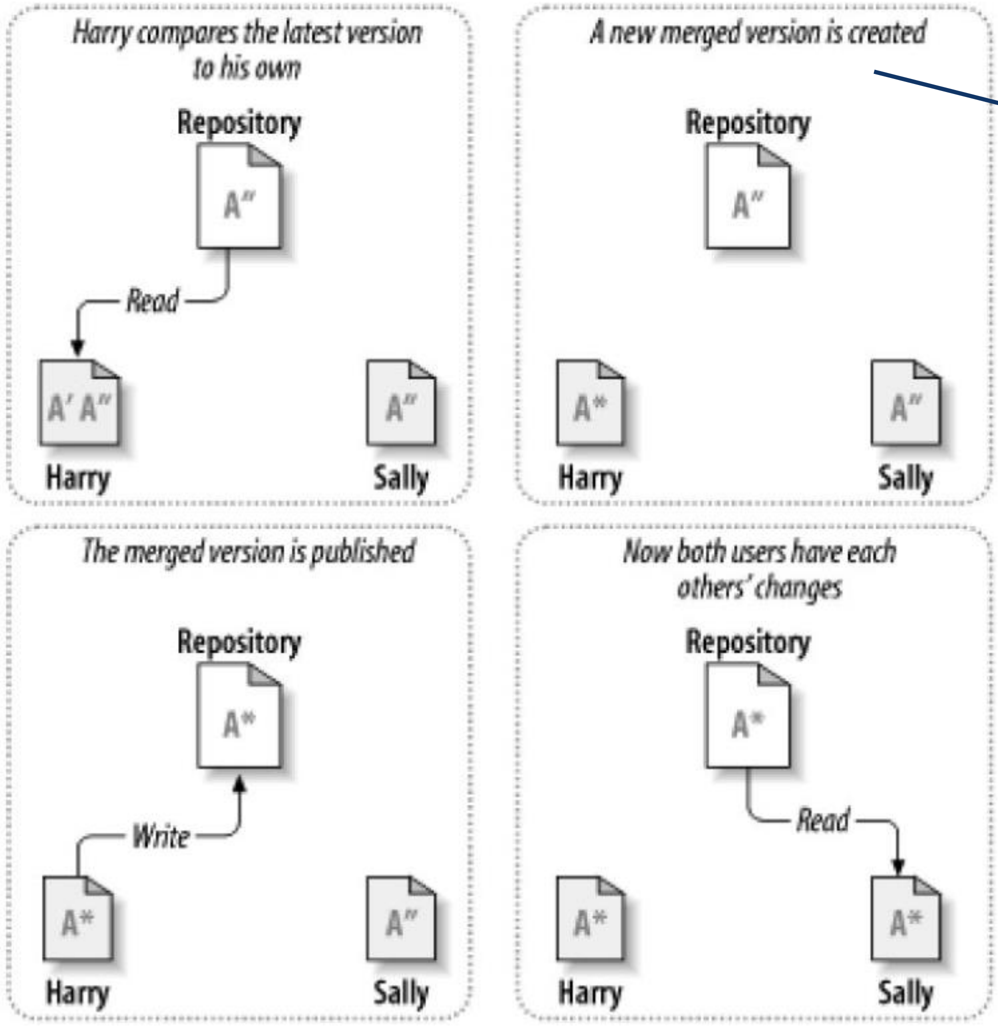

Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally

Source: SVN Help

Solution 1: The Merge

- In summary: copy-modify-merge
  - Each developer has unrestricted check-out access
  - Each developer edits the source and commits their changes asynchronously to the repository
  - The version control system performs `diff` operations on the versions committed to the repository and identifies *conflicts*
- Conflict resolution still requires human intervention
  - Sorry, but this doesn't mean you can stop talking to your colleagues – communication remains key
  - "Conflict resolution" may thus be understood in a number of senses ☺

Solution 1: The Merge (2)


Harry compares the latest version to his own
Repository
A″
Read
A′ A″
Harry
A″
Sally
A new merged version is created
Repository
A″
A*
Harry
A″
Sally
The merged version is published
Repository
A*
Write
A*
Harry
A″
Sally
Now both users have each others' changes
Repository
A*
Read
A*
Harry
A*
Sally

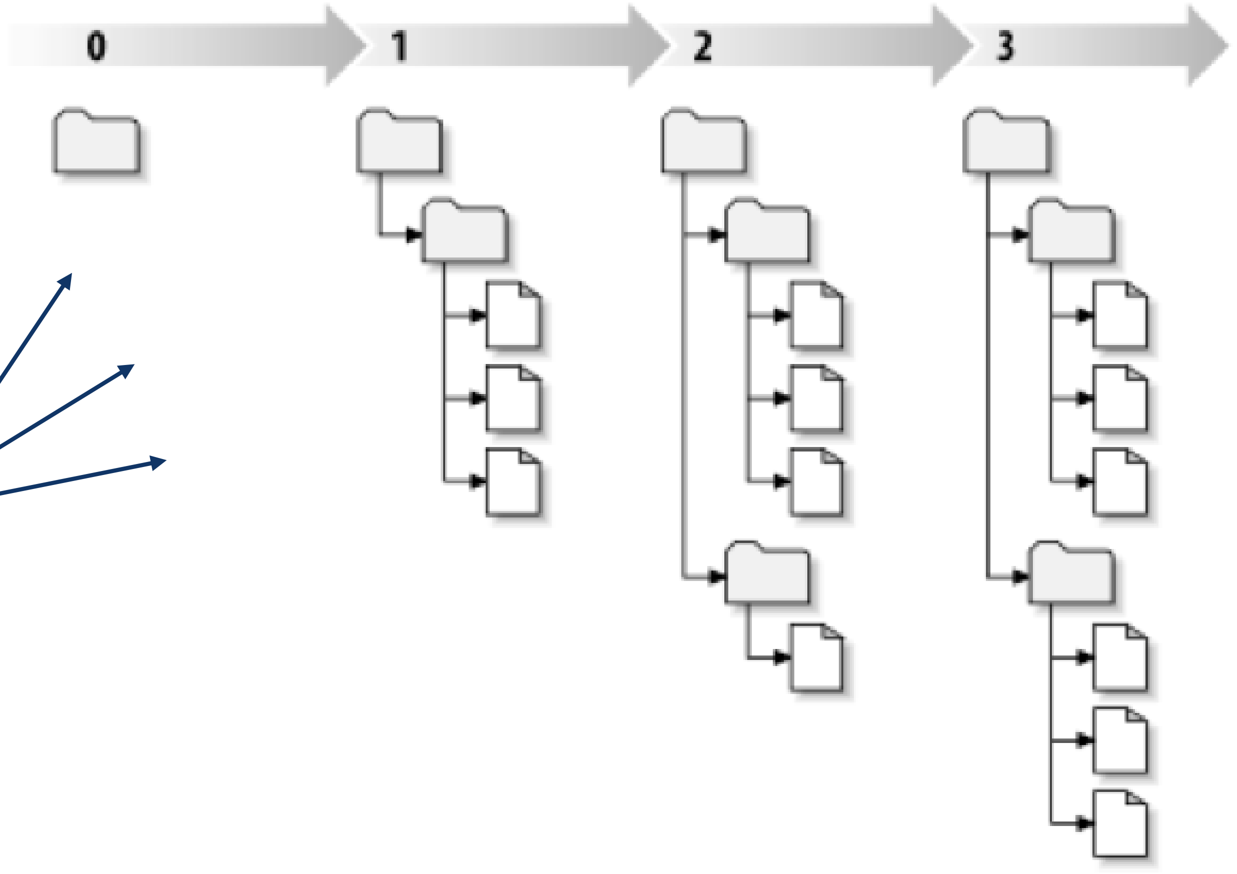
Harry and Sally need to agree on the merge!

Source: SVN Help

Version control with SVN

- Subversion (SVN) is one of many version control systems available
- SVN is the dominant centralised open source SCM system
  - CVS was for a long time the major player, but it's ugly and awkward
  - Like `make`, CVS will wither and die in time
  - Like CVS, SVN will wither and die in time
- Other systems include:
  - Microsoft Team Foundation Server (TFVC or git)
  - (IBM) Rational Clear Case
  - Git and Mercurial (very different; see more later)

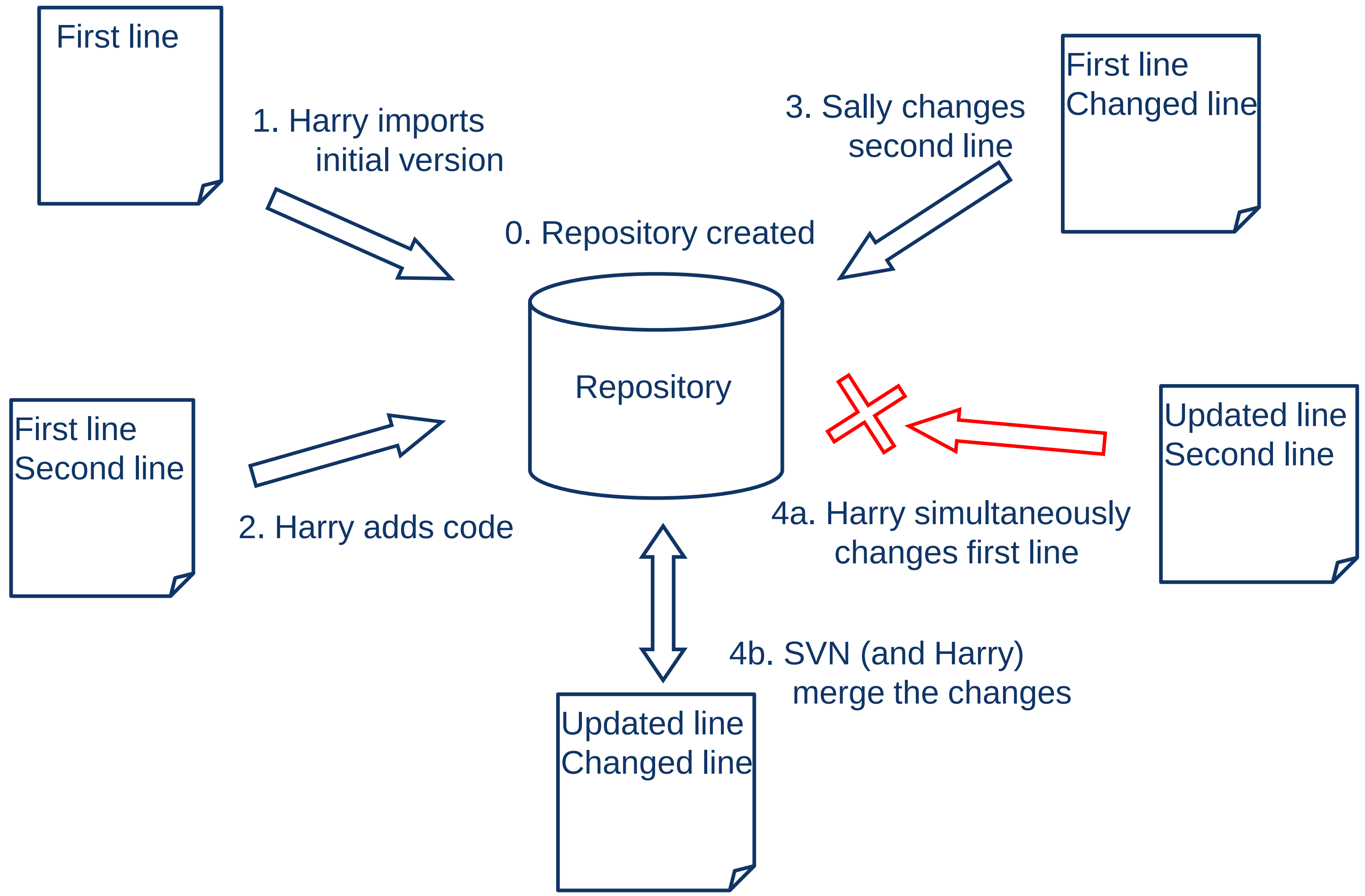
The repository



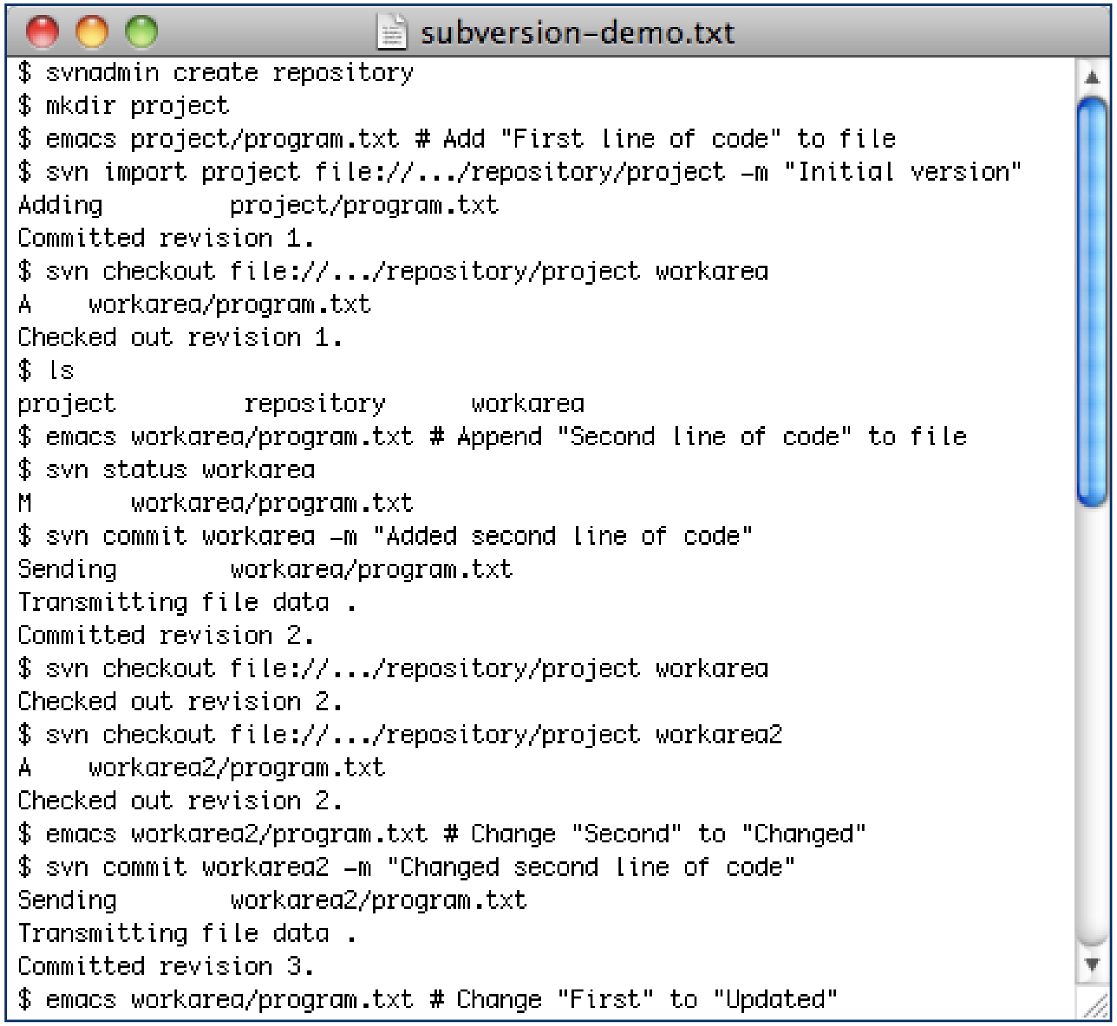
Different snapshots of the shared file system

Source: SVN Help

Subversion ‘demo’



Subversion (command line) demo

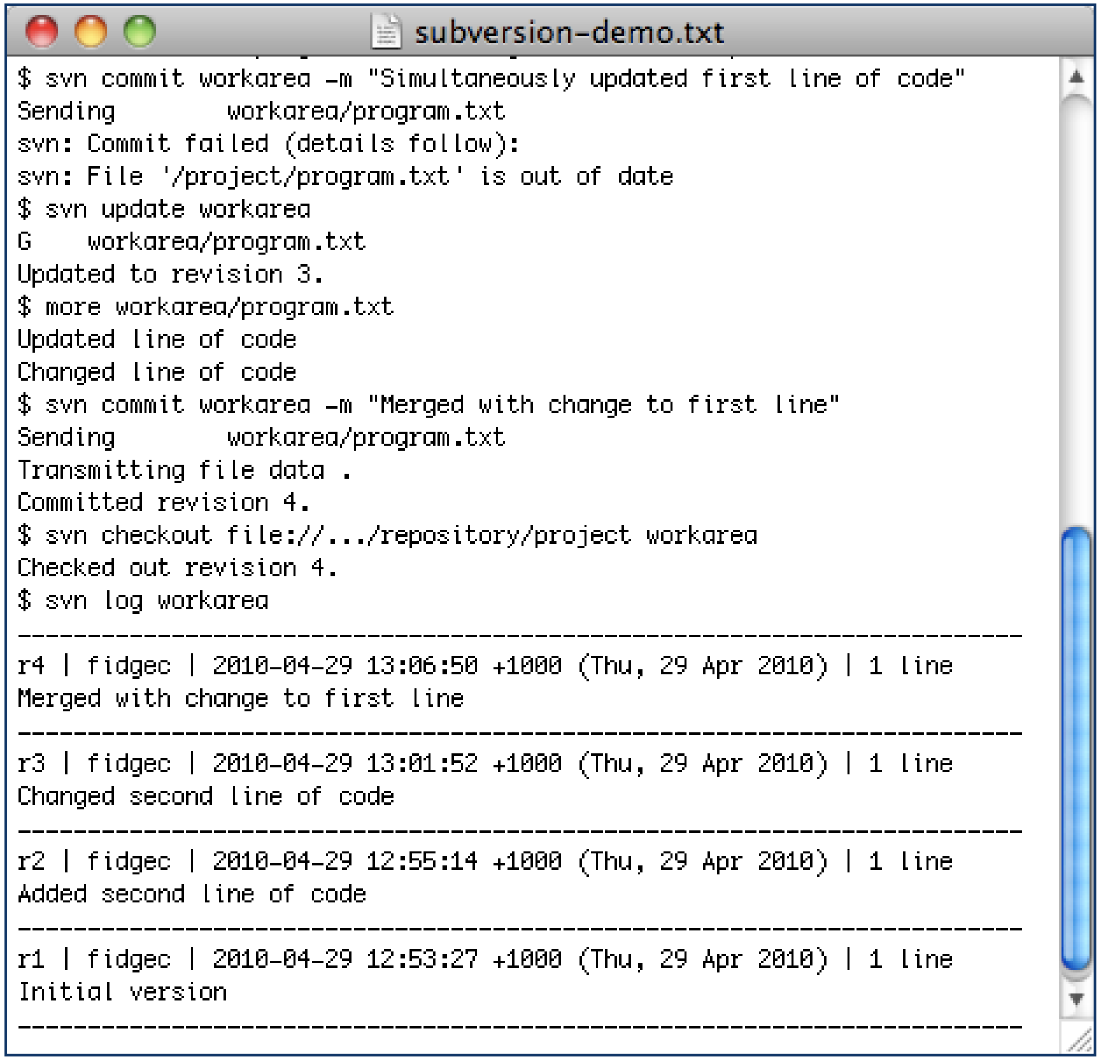


```
$ svnadmin create repository                                                  0
$ mkdir project
$ emacs project/program.txt # Add "First line of code" to file
$ svn import project file://.../repository/project -m "Initial version"
Adding         project/program.txt
Committed revision 1.                                                         1
$ svn checkout file://.../repository/project workarea
A    workarea/program.txt
Checked out revision 1.
$ ls
project          repository      workarea
$ emacs workarea/program.txt # Append "Second line of code" to file
$ svn status workarea
M       workarea/program.txt
$ svn commit workarea -m "Added second line of code"
Sending        workarea/program.txt
Transmitting file data .
Committed revision 2.                                                         2
$ svn checkout file://.../repository/project workarea
Checked out revision 2.
$ svn checkout file://.../repository/project workarea2
A    workarea2/program.txt
Checked out revision 2.
$ emacs workarea2/program.txt # Change "Second" to "Changed"
$ svn commit workarea2 -m "Changed second line of code"
Sending        workarea2/program.txt
Transmitting file data .
Committed revision 3.                                                         3
$ emacs workarea/program.txt # Change "First" to "Updated"
```

Subversion (command line) demo

4a

4b

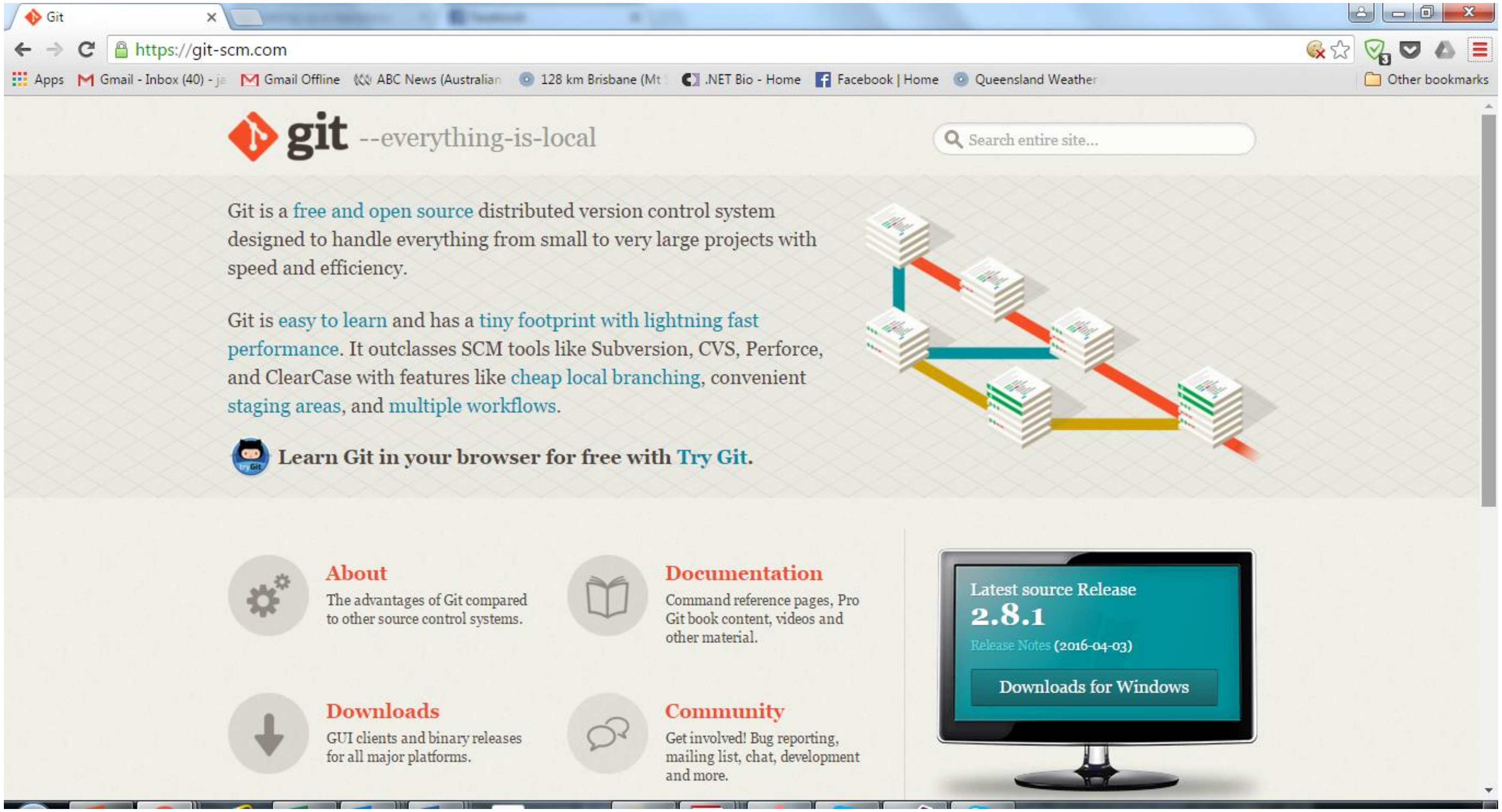


```
$ svn commit workarea -m "Simultaneously updated first line of code"
Sending        workarea/program.txt
svn: Commit failed (details follow):
svn: File '/project/program.txt' is out of date
$ svn update workarea
G    workarea/program.txt
Updated to revision 3.
$ more workarea/program.txt
Updated line of code
Changed line of code
$ svn commit workarea -m "Merged with change to first line"
Sending        workarea/program.txt
Transmitting file data .
Committed revision 4.
$ svn checkout file://.../repository/project workarea
Checked out revision 4.
$ svn log workarea
------------------------------------------------------------------------
r4 | fidgec | 2010-04-29 13:06:50 +1000 (Thu, 29 Apr 2010) | 1 line
Merged with change to first line
------------------------------------------------------------------------
r3 | fidgec | 2010-04-29 13:01:52 +1000 (Thu, 29 Apr 2010) | 1 line
Changed second line of code
------------------------------------------------------------------------
r2 | fidgec | 2010-04-29 12:55:14 +1000 (Thu, 29 Apr 2010) | 1 line
Added second line of code
------------------------------------------------------------------------
r1 | fidgec | 2010-04-29 12:53:27 +1000 (Thu, 29 Apr 2010) | 1 line
Initial version
------------------------------------------------------------------------
```

The Story so Far (Continued)

- We have now introduced a history of traditional version control with some focus on the issues that arise with a central repo
- SVN is the best example for modern centralised source control
- But the world is moving toward a distributed model.
- Centralised servers have some disadvantages:
  - Central point of failure
  - Latency if remote
  - Inaccessibility if off-line, making team work unnecessarily sequential rather than parallel, introducing conflicts
  - (Same for local repos of course)
- Distributed version control means multiple copies…

GIT



Git

https://git-scm.com

Apps | Gmail - Inbox (40) - ja | Gmail Offline | ABC News (Australian | 128 km Brisbane (Mt | .NET Bio - Home | Facebook | Home | Queensland Weather | Other bookmarks

git --everything-is-local

Search entire site...

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Learn Git in your browser for free with Try Git.

About
The advantages of Git compared to other source control systems.

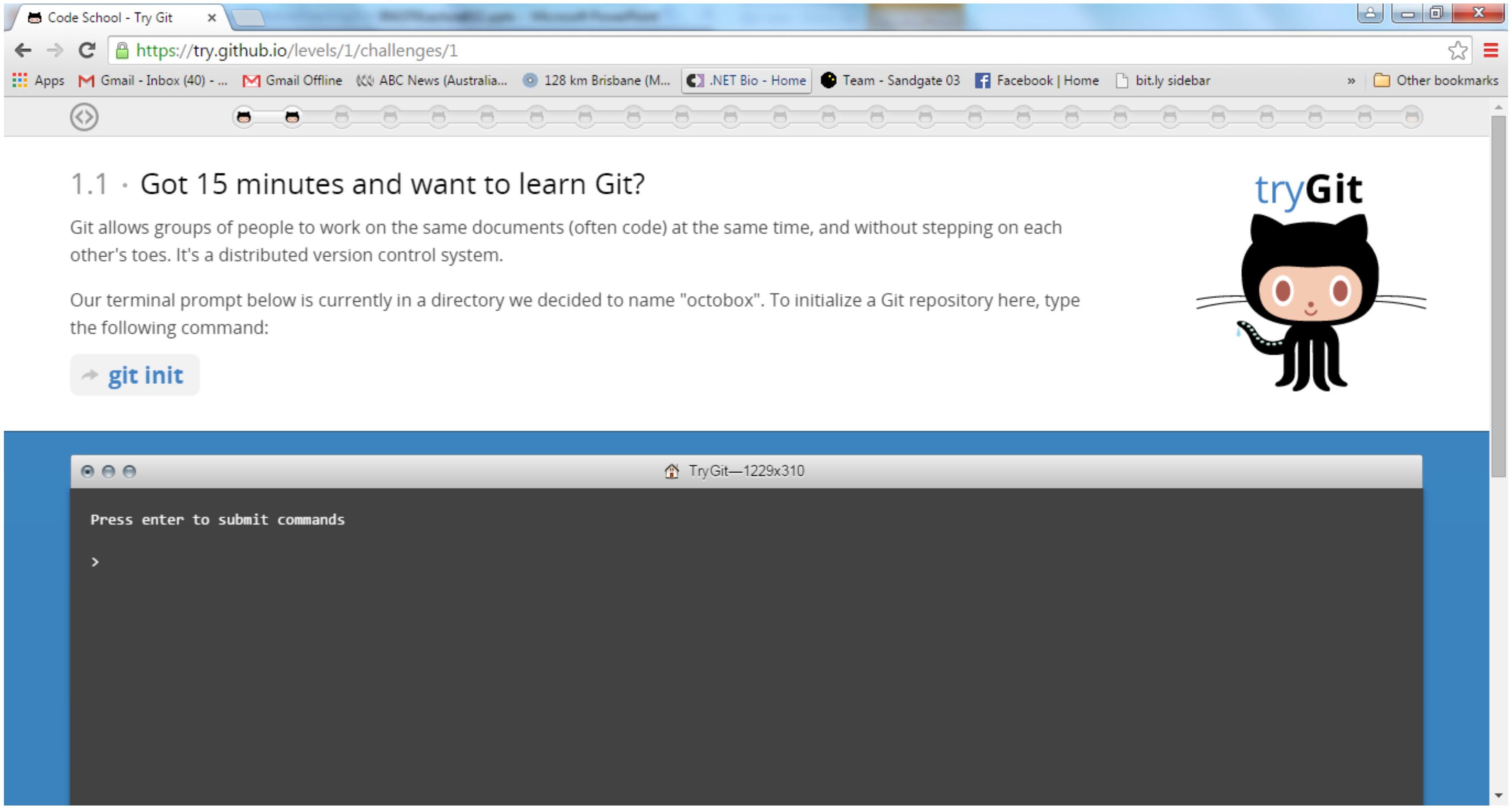
Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.8.1
Release Notes (2016-04-03)
Downloads for Windows

http://try.github.com/



1.1 · Got 15 minutes and want to learn Git?

Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

Our terminal prompt below is currently in a directory we decided to name "octobox". To initialize a Git repository here, type the following command:

```
git init
```

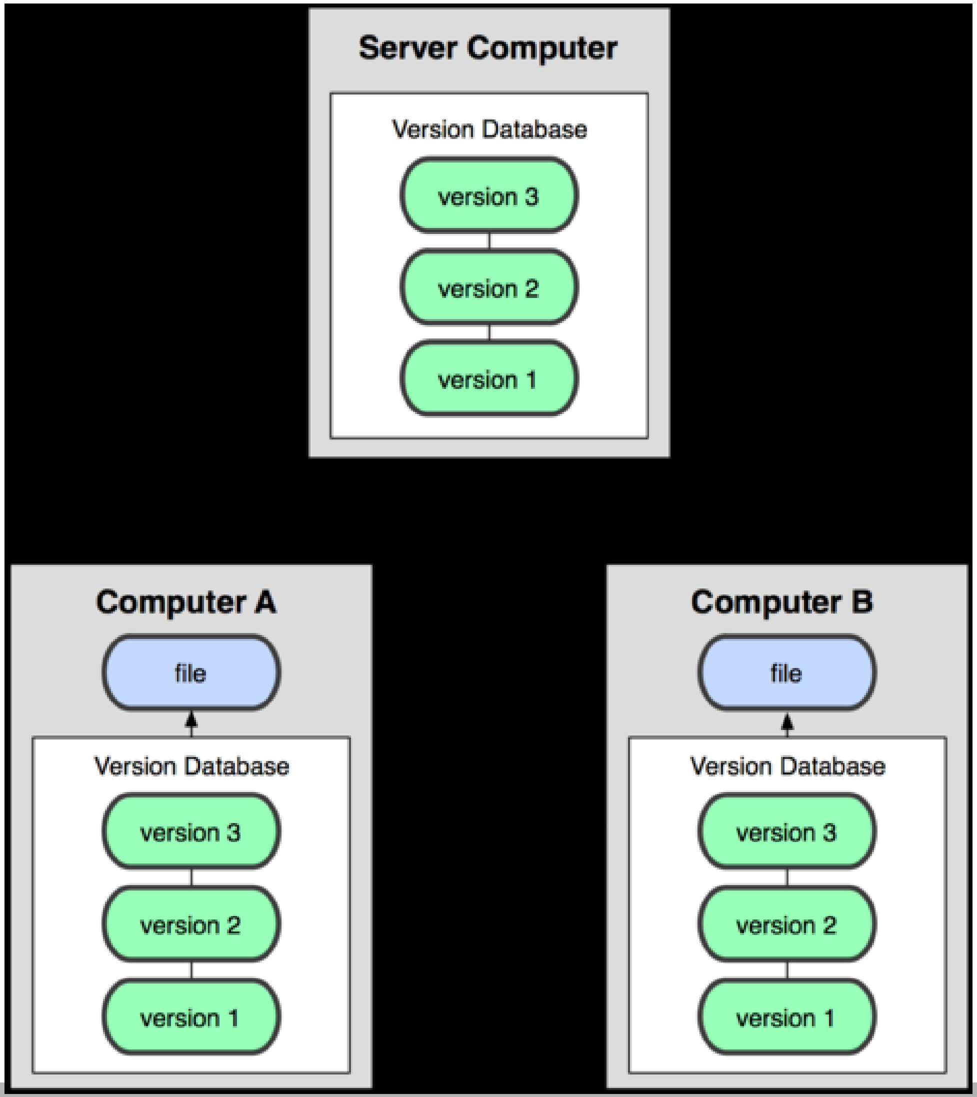
```
Press enter to submit commands

>
```

Version Control with Git

- With Git, everything is local
- SVN et al rely on a defined central repository
- With Git, everyone has a copy of the whole repository, and a history of how it got to be that way.
- So, updates are much faster as there is less network use
- Greater flexibility in merging and branching

- Main danger with Git is trap of having only one repository,
  - A danger not restricted to Git
- Use a local one on your machine, and then get one from Bitbucket or Github or set one up on another server.

Distributed Version Control



Note the difference – we don't just check out the current version of the file we are working on, we are maintaining a copy of the entire repo

This means that we have very low latency for file inspection, copy and comparisons.

The idea of a file lock is essentially meaningless

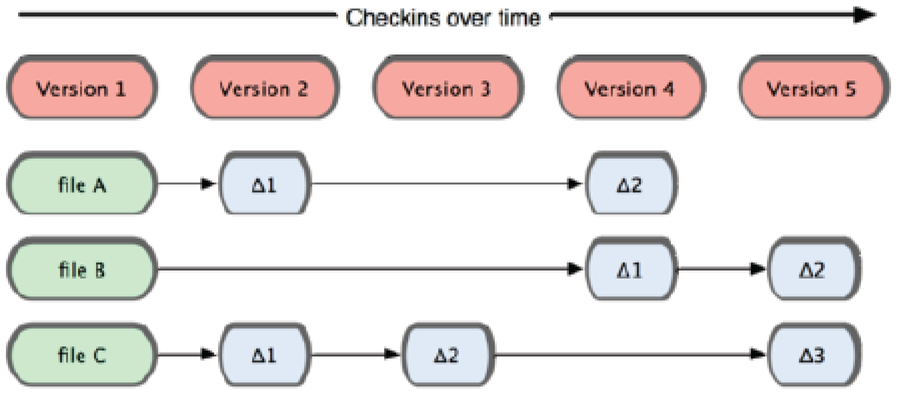
But we still have the file merge to consider.

DVCS History

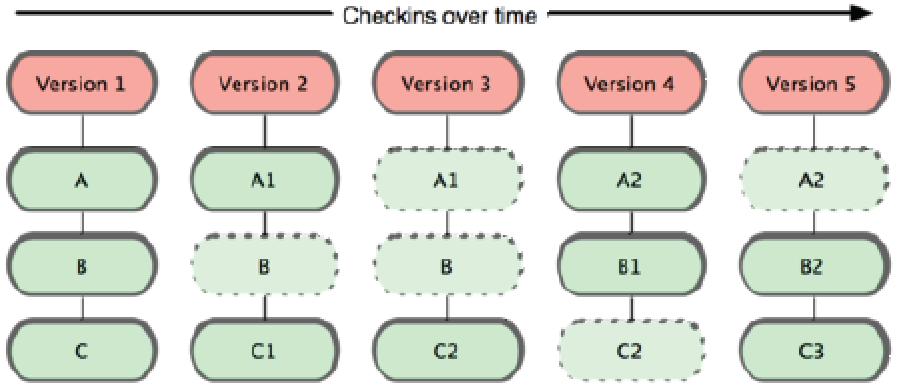
- Bitkeeper (early 2000s) used in the Linux project via a free license
- As with Harry and Sally's commits, this didn't end well
- Linus Torvalds led development of a new DVCS with goals:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle the Linux project scale (D'Oh!)
- Fundamentally different view of the project data
  - Earlier: series of diffs from original version
  - Git: series of whole snapshots

Repo Vision

RCS, CVS, SVN view



Git view

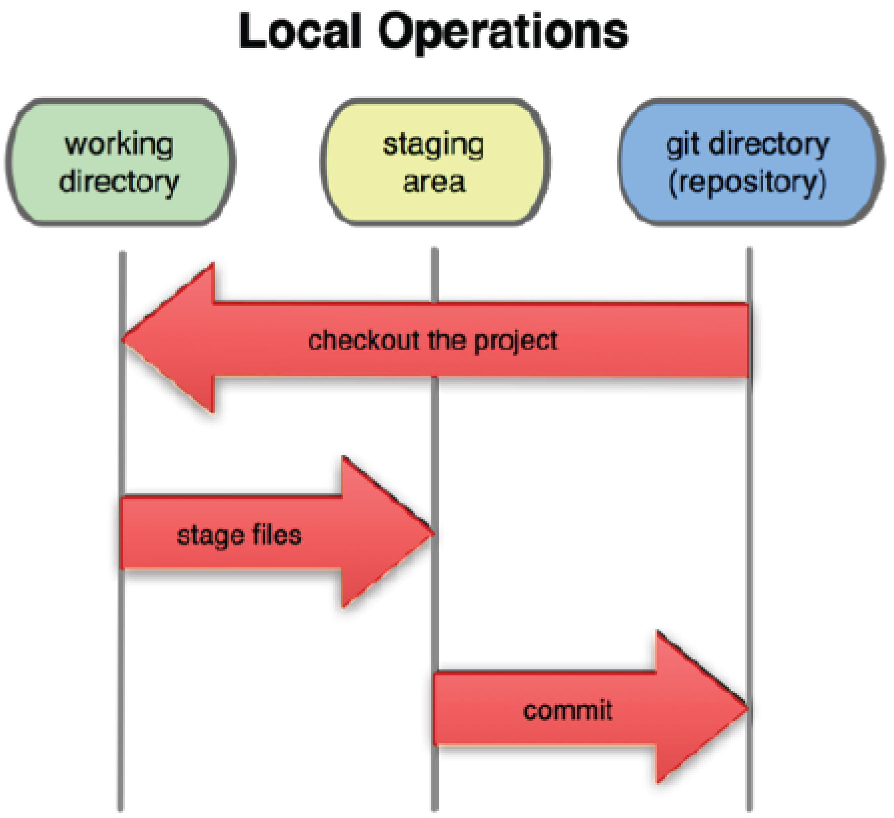


The Git Worldview

Git directory – repo and project metadata

Working directory – checkout of the latest version of the project. May contain files you have modified, but not staged or committed.

Staging area – file indicating the modified files in the working directory to be included in the next commit.

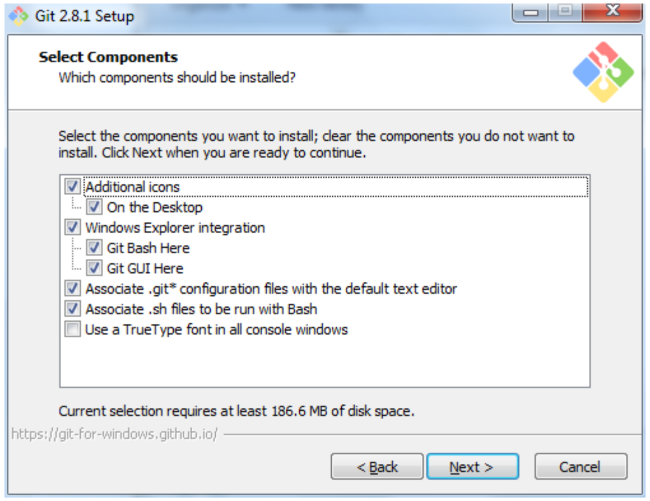


Working with Git

- For the remainder of the lecture we will work with some very basic local and remote Git operations.
- These are crucial to understanding the idea of Source Control and working with a repository in practice
- Mostly our work is drawn from Chapter 2 of the Git Book – we make here only limited reference to branching. While this is crucial, we leave it to the prac as there is enough to deal with here.
- We begin with Git as a local installation and focus on a local repository.
- Over time, we will create another repository by cloning one from Bitbucket

The Git Install

- Install options matter
  - Explorer integration
  - Command line tools
  - CR/LF handling



Git 2.8.1 Setup

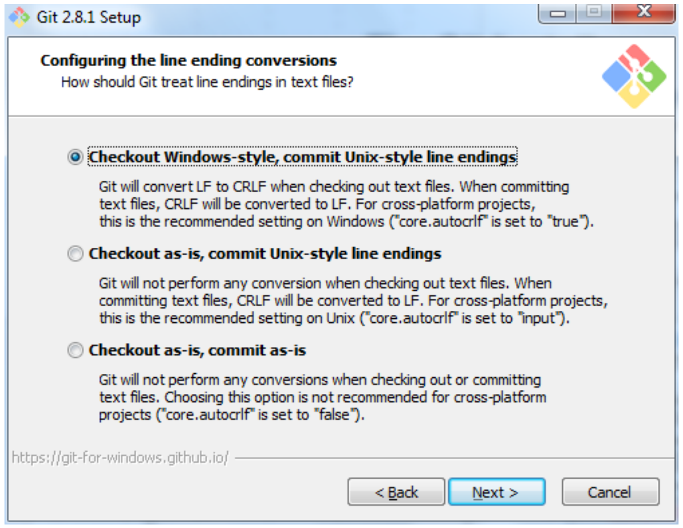
Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows

Current selection requires at least 186.6 MB of disk space.

https://git-for-windows.github.io/



Git 2.8.1 Setup

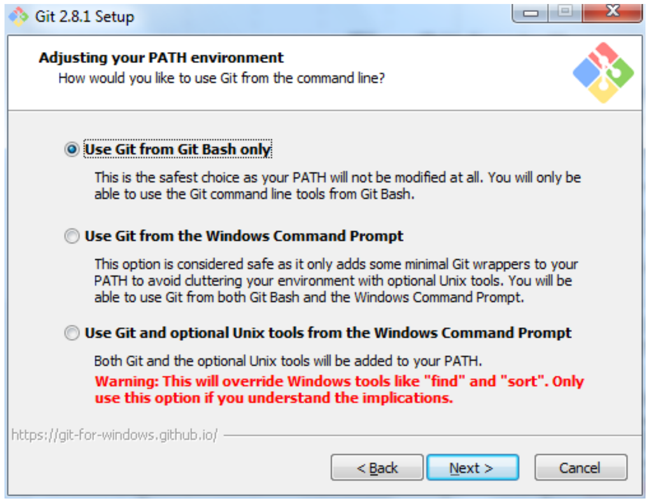
Configuring the line ending conversions
How should Git treat line endings in text files?

(●) **Checkout Windows-style, commit Unix-style line endings**
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

( ) **Checkout as-is, commit Unix-style line endings**
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

( ) **Checkout as-is, commit as-is**
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

https://git-for-windows.github.io/



Git 2.8.1 Setup

Adjusting your PATH environment
How would you like to use Git from the command line?

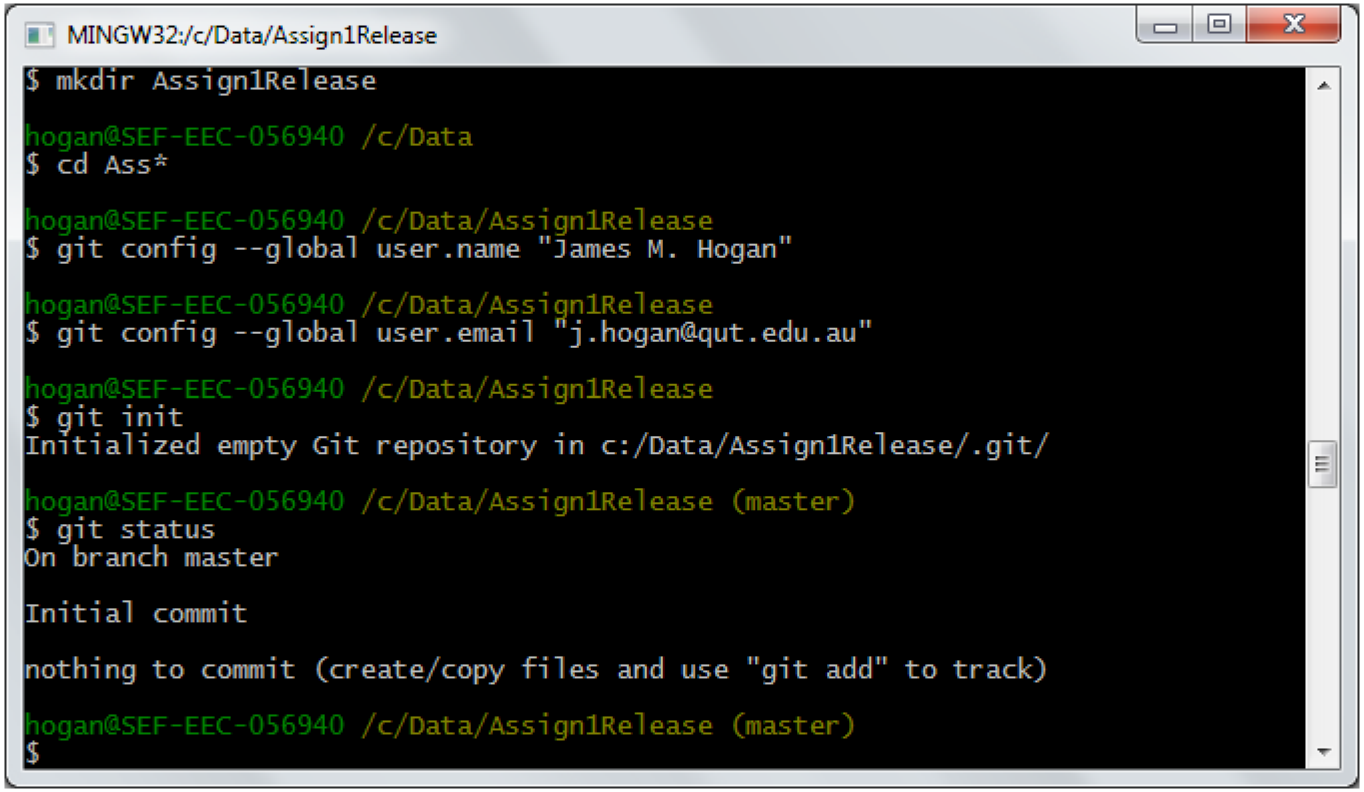
(●) **Use Git from Git Bash only**
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

( ) **Use Git from the Windows Command Prompt**
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

( ) **Use Git and optional Unix tools from the Windows Command Prompt**
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://git-for-windows.github.io/

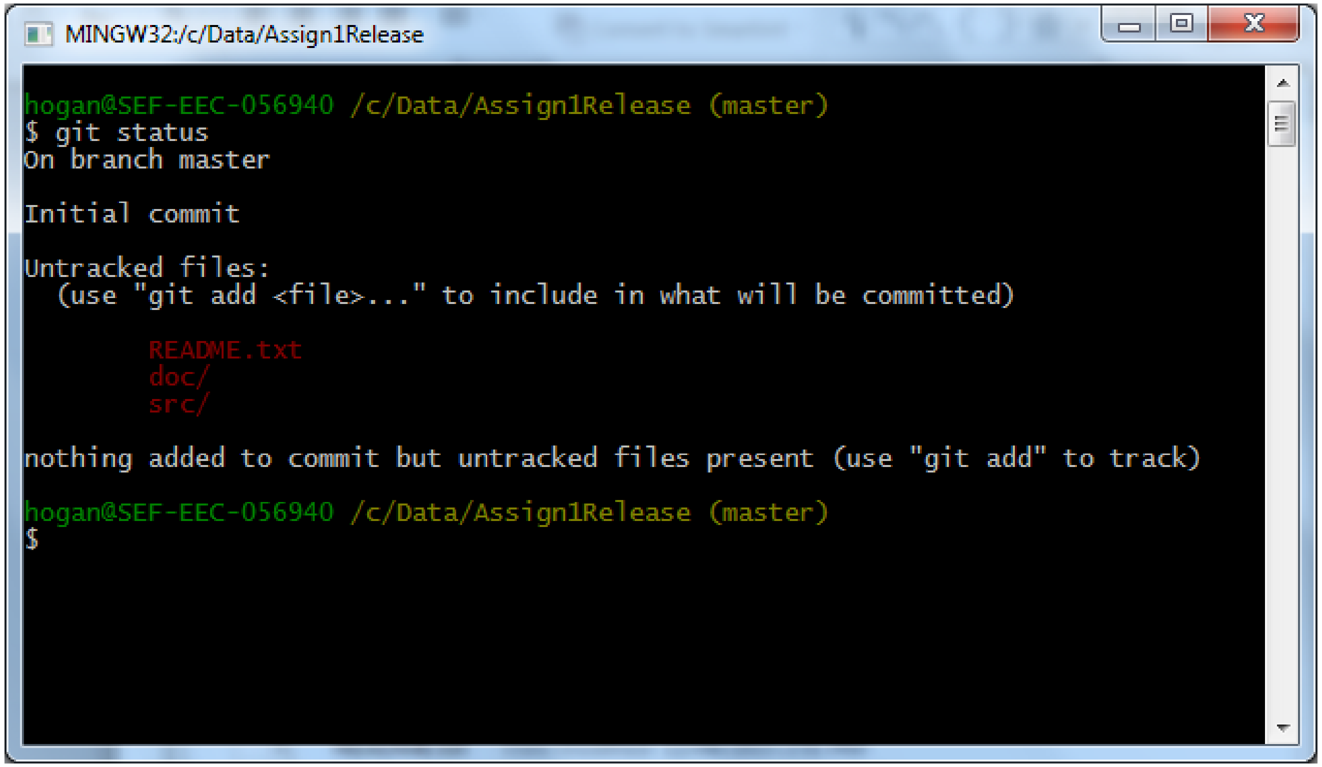
Getting Started



```
$ mkdir Assign1Release

hogan@SEF-EEC-056940 /c/Data
$ cd Ass*

hogan@SEF-EEC-056940 /c/Data/Assign1Release
$ git config --global user.name "James M. Hogan"

hogan@SEF-EEC-056940 /c/Data/Assign1Release
$ git config --global user.email "j.hogan@qut.edu.au"

hogan@SEF-EEC-056940 /c/Data/Assign1Release
$ git init
Initialized empty Git repository in c:/Data/Assign1Release/.git/

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$
```

Assign1Release is an empty directory
I have set up my name and email, and initialised the empty repo
Status tells us that no files are yet available for commit

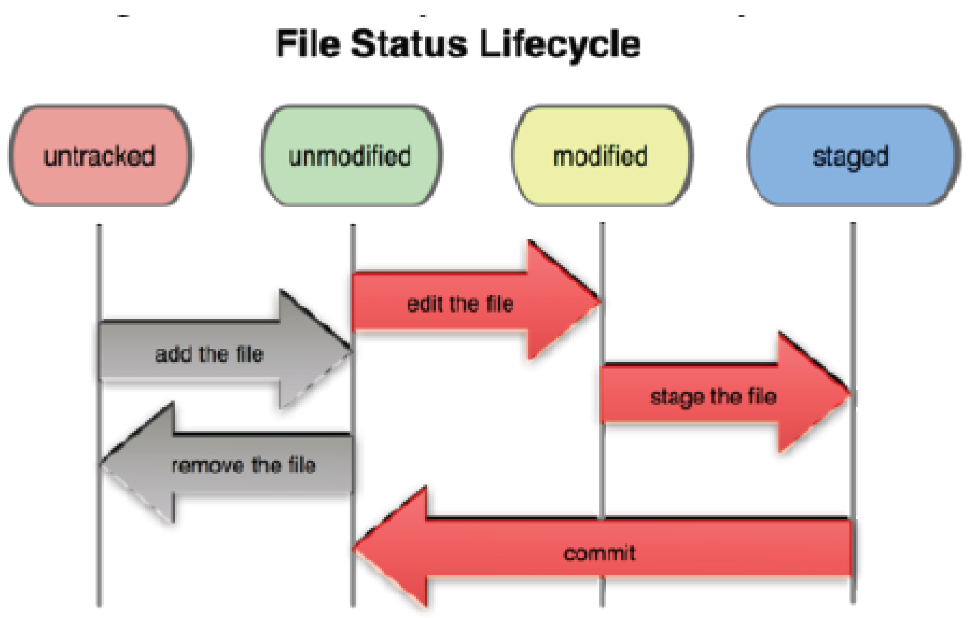
Getting Started


MINGW32:/c/Data/Assign1Release

```
hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.txt
        doc/
        src/

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

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$
```

I have copied into the file system under Assign1Release the src and doc directories, and a dummy README file.
Git status tells us we have untracked files – files in the working directory unknown to Git
We make them available by using Git add – this is called staging

The Git Lifecycle



Note how *git commit* restores the status to unmodified

The working directory is a mix of files: most unchanged from the repo, some modified, some not tracked

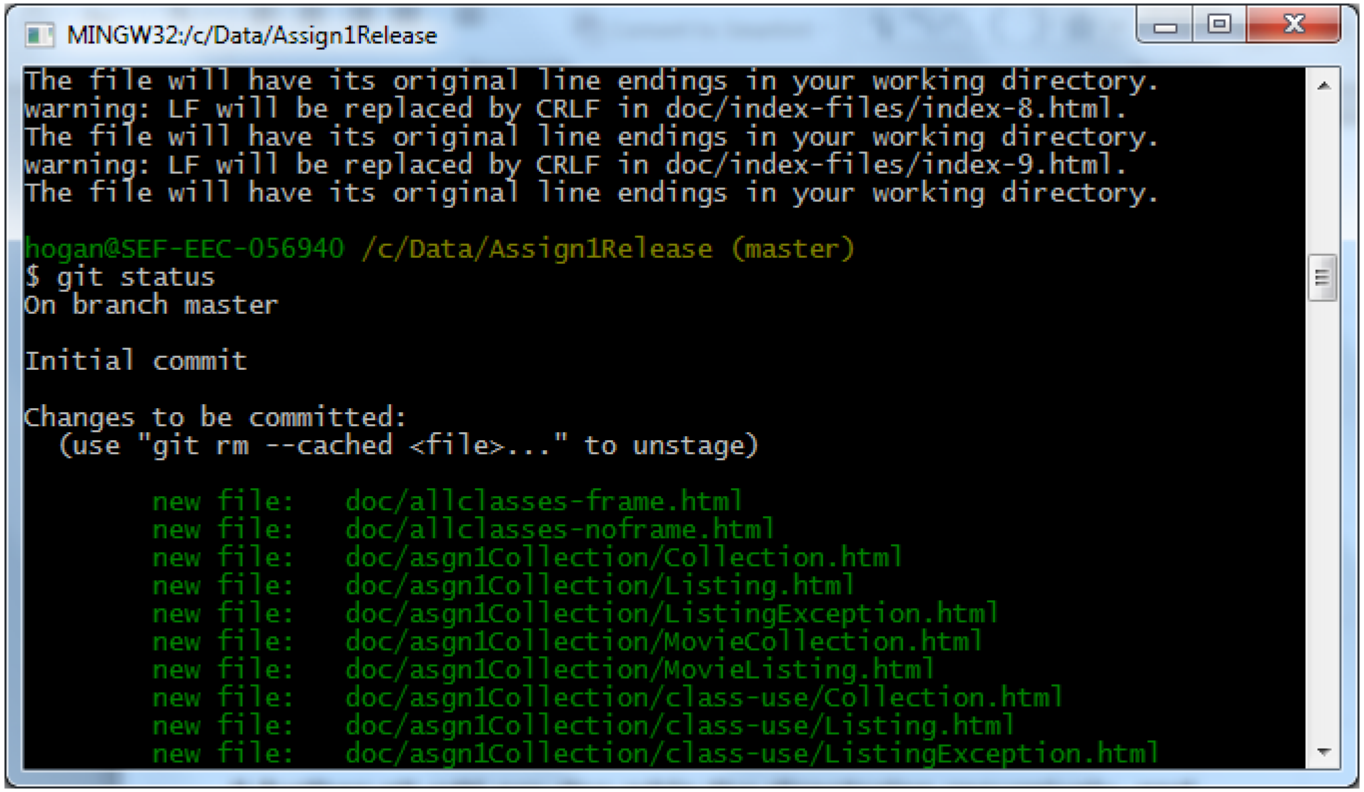
Git uses checksums to tell whether the files have changed.

git add is overloaded

- Tells git to track files
- Tells git to stage tracked files that have been modified.
- (this applies recursively to directories of course)

git commit commits staged files

Getting Started

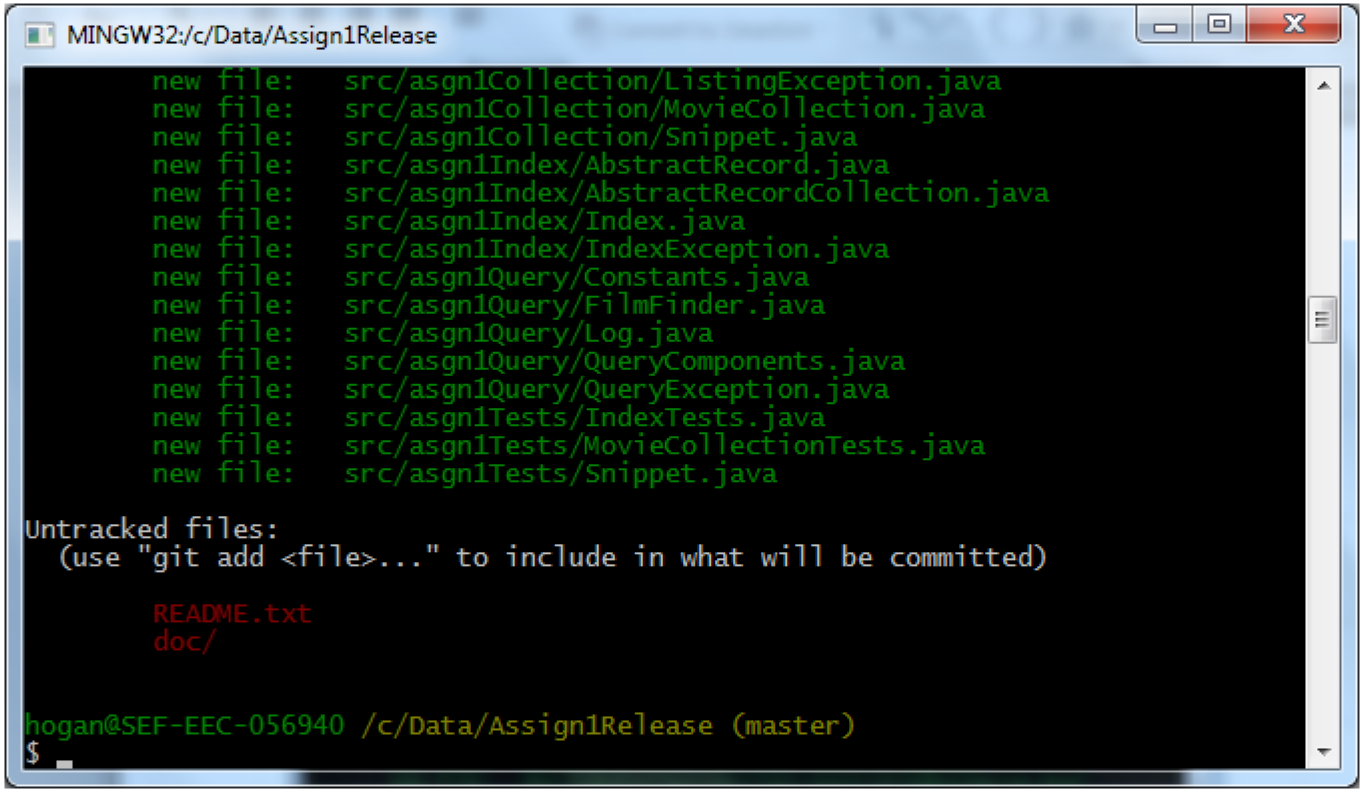


MINGW32:/c/Data/Assign1Release

```
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in doc/index-files/index-8.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in doc/index-files/index-9.html.
The file will have its original line endings in your working directory.

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   doc/allclasses-frame.html
        new file:   doc/allclasses-noframe.html
        new file:   doc/asgn1Collection/Collection.html
        new file:   doc/asgn1Collection/Listing.html
        new file:   doc/asgn1Collection/ListingException.html
        new file:   doc/asgn1Collection/MovieCollection.html
        new file:   doc/asgn1Collection/MovieListing.html
        new file:   doc/asgn1Collection/class-use/Collection.html
        new file:   doc/asgn1Collection/class-use/Listing.html
        new file:   doc/asgn1Collection/class-use/ListingException.html
```

A *git add src doc* adds the directories recursively, and generates lots of warnings about line feeds.
The status shows the large number of files now in the staging area.

Changing Your Mind

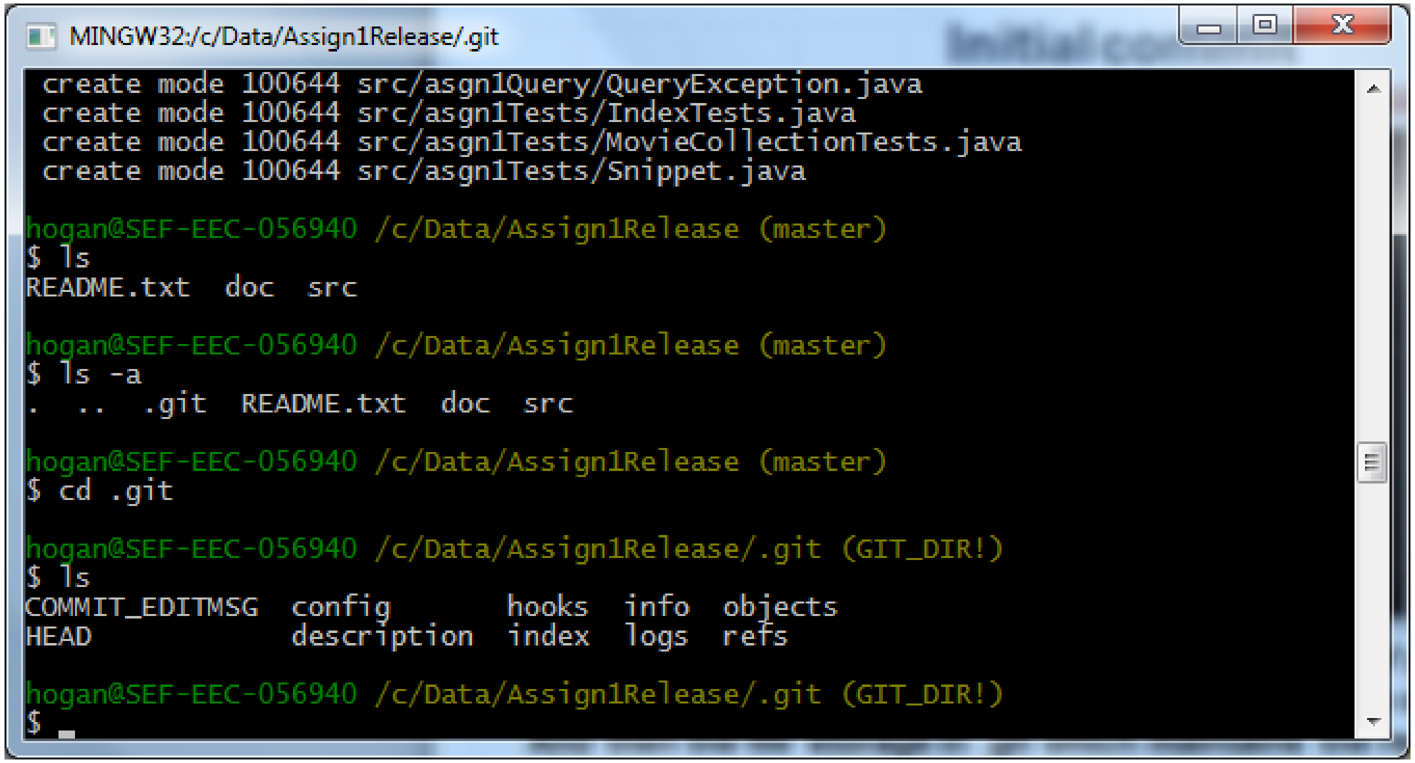


MINGW32:/c/Data/Assign1Release

```
        new file:   src/asgn1Collection/ListingException.java
        new file:   src/asgn1Collection/MovieCollection.java
        new file:   src/asgn1Collection/Snippet.java
        new file:   src/asgn1Index/AbstractRecord.java
        new file:   src/asgn1Index/AbstractRecordCollection.java
        new file:   src/asgn1Index/Index.java
        new file:   src/asgn1Index/IndexException.java
        new file:   src/asgn1Query/Constants.java
        new file:   src/asgn1Query/FilmFinder.java
        new file:   src/asgn1Query/Log.java
        new file:   src/asgn1Query/QueryComponents.java
        new file:   src/asgn1Query/QueryException.java
        new file:   src/asgn1Tests/IndexTests.java
        new file:   src/asgn1Tests/MovieCollectionTests.java
        new file:   src/asgn1Tests/Snippet.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.txt
        doc/


hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ _
```

Add is recursive; remove is not. The syntax is careful. You either keep the file (--cached) or you don’t (-f). Command to unstage doc: *git rm --cached –r doc* [Note that doc is now untracked again]. We can also checkout over the top of files in the workspace.

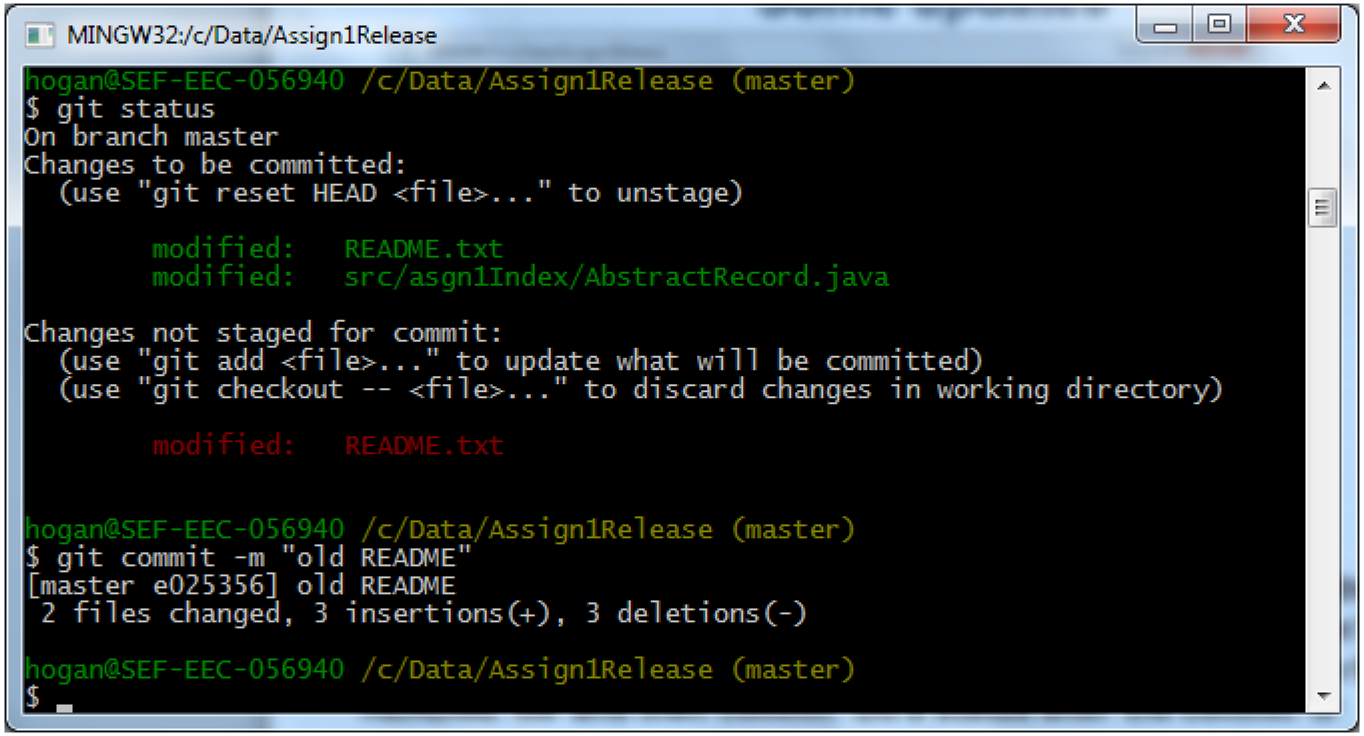
Initial commit


MINGW32:/c/Data/Assign1Release/.git

```
 create mode 100644 src/asgn1Query/QueryException.java
 create mode 100644 src/asgn1Tests/IndexTests.java
 create mode 100644 src/asgn1Tests/MovieCollectionTests.java
 create mode 100644 src/asgn1Tests/Snippet.java

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ ls
README.txt  doc  src

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ ls -a
.  ..  .git  README.txt  doc  src

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ cd .git

hogan@SEF-EEC-056940 /c/Data/Assign1Release/.git (GIT_DIR!)
$ ls
COMMIT_EDITMSG  config       hooks  info  objects
HEAD            description  index  logs  refs

hogan@SEF-EEC-056940 /c/Data/Assign1Release/.git (GIT_DIR!)
$ _
```

Commands here: *git add .* [stages everything] and then *commit –m "Initial commit"* Note the creation of files in the actual repository (top of screen). And then the file storage in .git which maintains the repository and all the associated metadata. Use git to manage this. DON'T edit

Some updates

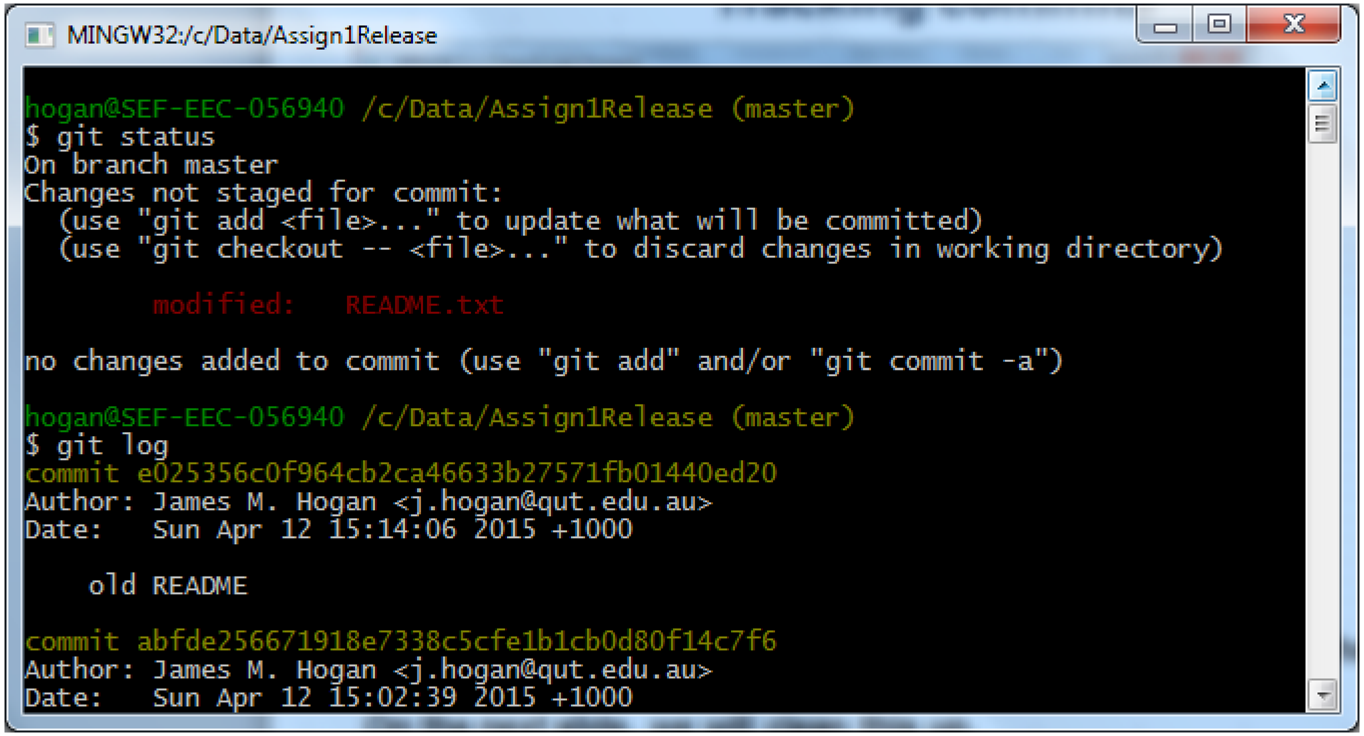


MINGW32:/c/Data/Assign1Release

```
hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        modified:   src/asgn1Index/AbstractRecord.java

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:   README.txt


hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git commit -m "old README"
[master e025356] old README
 2 files changed, 3 insertions(+), 3 deletions(-)

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ _
```

I have now gone and edited the README and a Java file from the src. The git status command tells me that these changes haven't been staged, so we add. Crucial point: I now make a change to the README file and then commit. Do a status after the commit ☺

Tracking commits

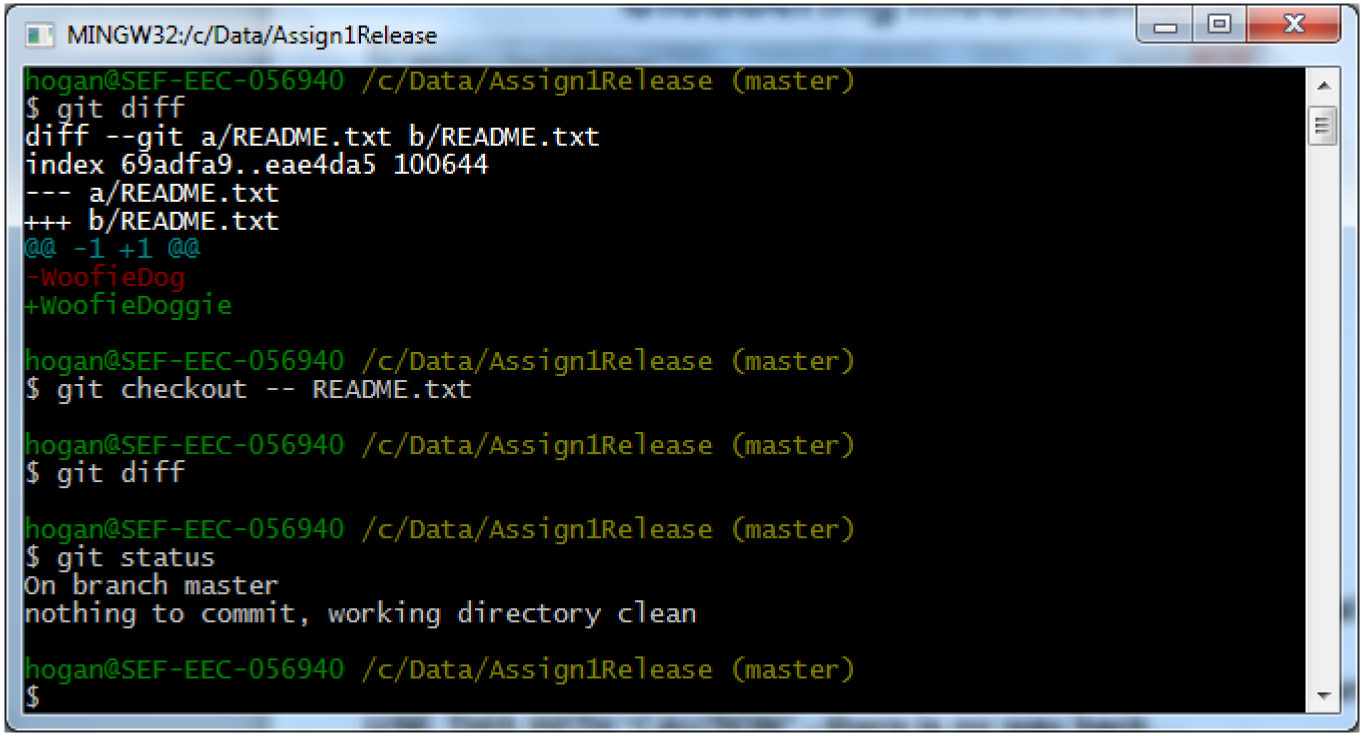


MINGW32:/c/Data/Assign1Release

```
hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ 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:   README.txt

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

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git log
commit e025356c0f964cb2ca46633b27571fb01440ed20
Author: James M. Hogan <j.hogan@qut.edu.au>
Date:   Sun Apr 12 15:14:06 2015 +1000

    old README

commit abfde256671918e7338c5cfe1b1cb0d80f14c7f6
Author: James M. Hogan <j.hogan@qut.edu.au>
Date:   Sun Apr 12 15:02:39 2015 +1000
```

The status shows the file is inconsistent
The *git log* command gives a basic summary of the commits that have been made.
On the next slide, we will clean this up.

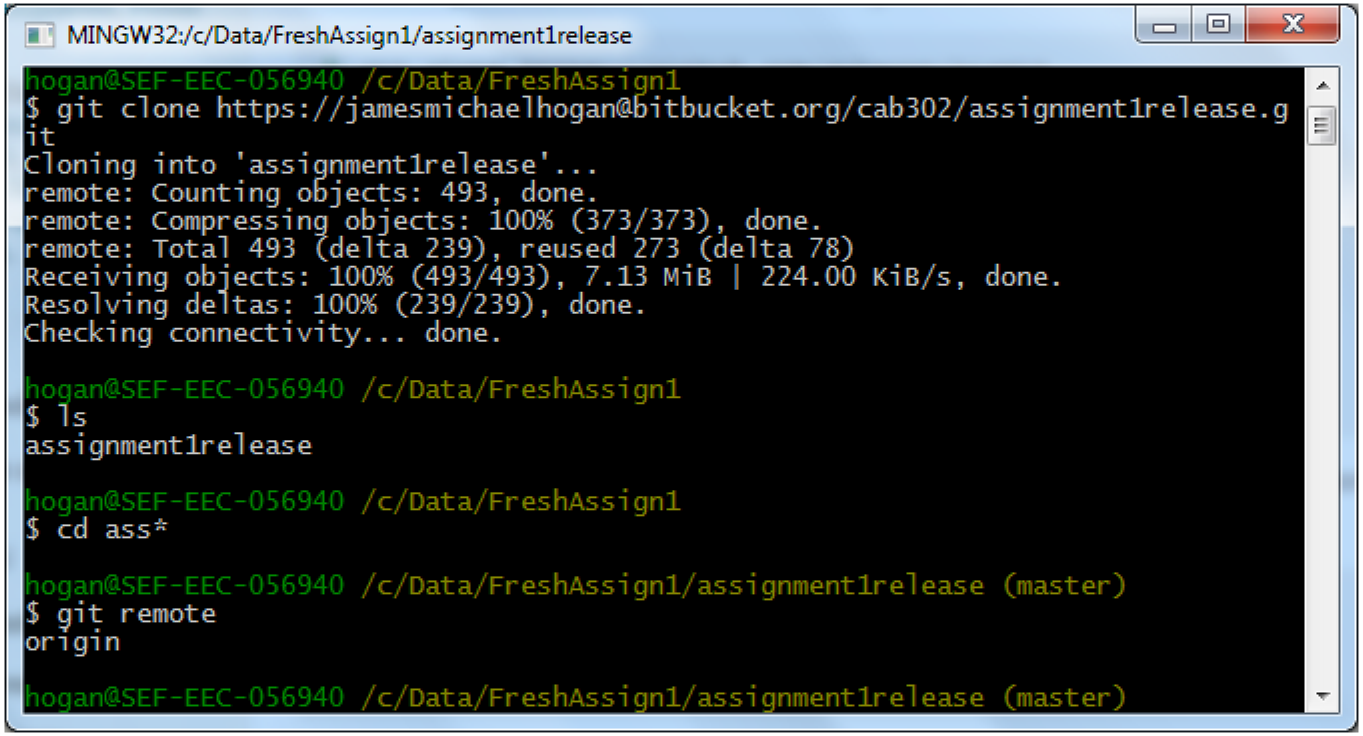
Clobbering modifications


MINGW32:/c/Data/Assign1Release

```
hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git diff
diff --git a/README.txt b/README.txt
index 69adfa9..eae4da5 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-WoofieDog
+WoofieDoggie

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git checkout -- README.txt

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git diff

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$ git status
On branch master
nothing to commit, working directory clean

hogan@SEF-EEC-056940 /c/Data/Assign1Release (master)
$
```

Simple. The diff shows that we have an issue with an additional line in the modified file. We clobber this modified file by checkout from the repo and the working directory is clean.
USE THIS WITH *CAUTION* - there is no way back…

Cloning a remote repository

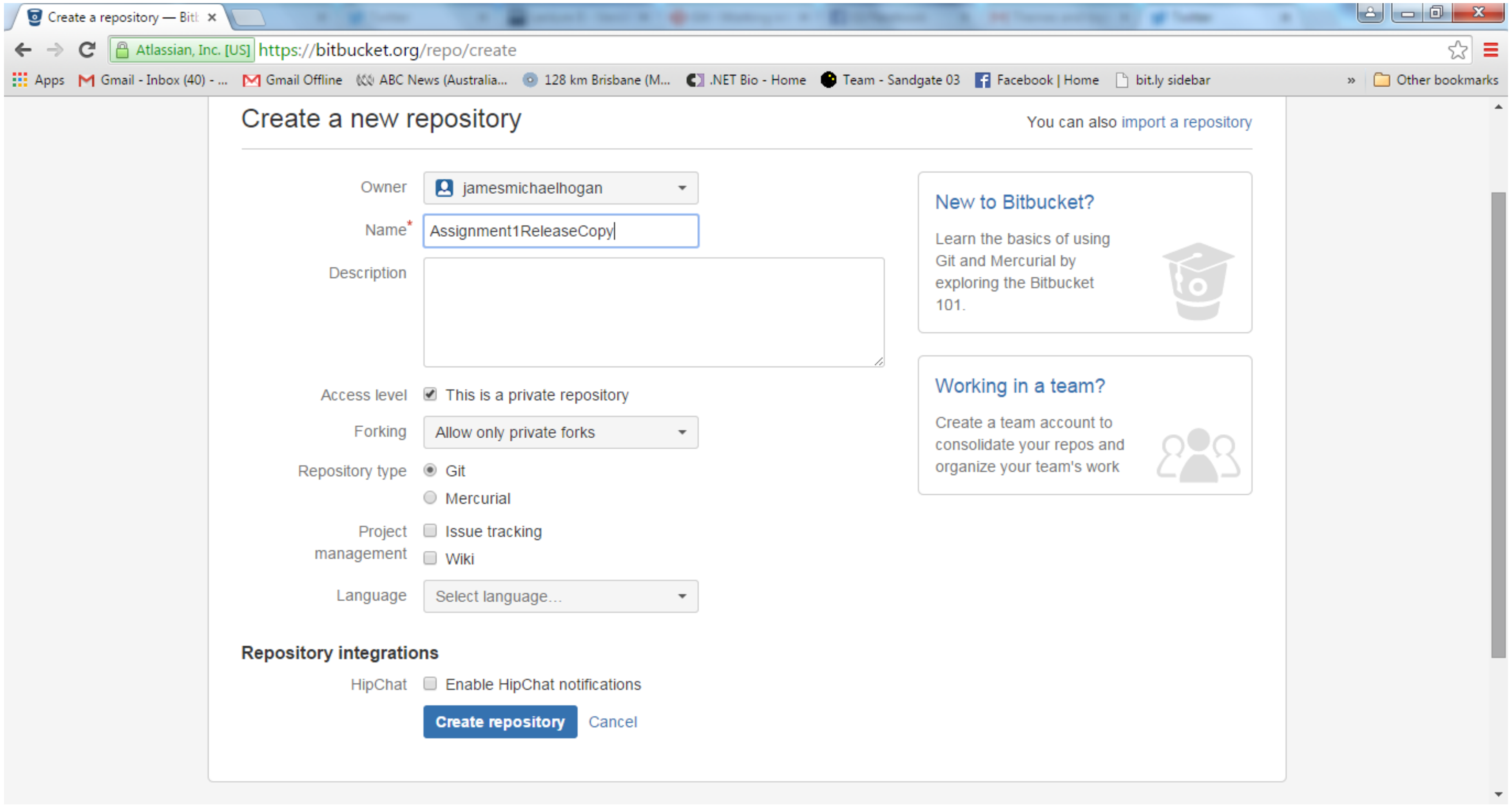


MINGW32:/c/Data/FreshAssign1/assignment1release

```
hogan@SEF-EEC-056940 /c/Data/FreshAssign1
$ git clone https://jamesmichaelhogan@bitbucket.org/cab302/assignment1release.g
it
Cloning into 'assignment1release'...
remote: Counting objects: 493, done.
remote: Compressing objects: 100% (373/373), done.
remote: Total 493 (delta 239), reused 273 (delta 78)
Receiving objects: 100% (493/493), 7.13 MiB | 224.00 KiB/s, done.
Resolving deltas: 100% (239/239), done.
Checking connectivity... done.

hogan@SEF-EEC-056940 /c/Data/FreshAssign1
$ ls
assignment1release

hogan@SEF-EEC-056940 /c/Data/FreshAssign1
$ cd ass*

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git remote
origin

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
```

A fresh copy of the assignment 1 release. Cloned and sent to my local machine using https. Note the remote shortname ‘origin’. [Bitbucket is pretty strict on use of secure channels]

Create a repo



Create a new repository

You can also import a repository

Owner	jamesmichaelhogan
Name*	Assignment1ReleaseCopy
Description	
Access level	[✓] This is a private repository
Forking	Allow only private forks
Repository type	[✓ Git] Mercurial
Project management	[ ] Issue tracking [ ] Wiki
Language	Select language...

Repository integrations

HipChat	[ ] Enable HipChat notifications

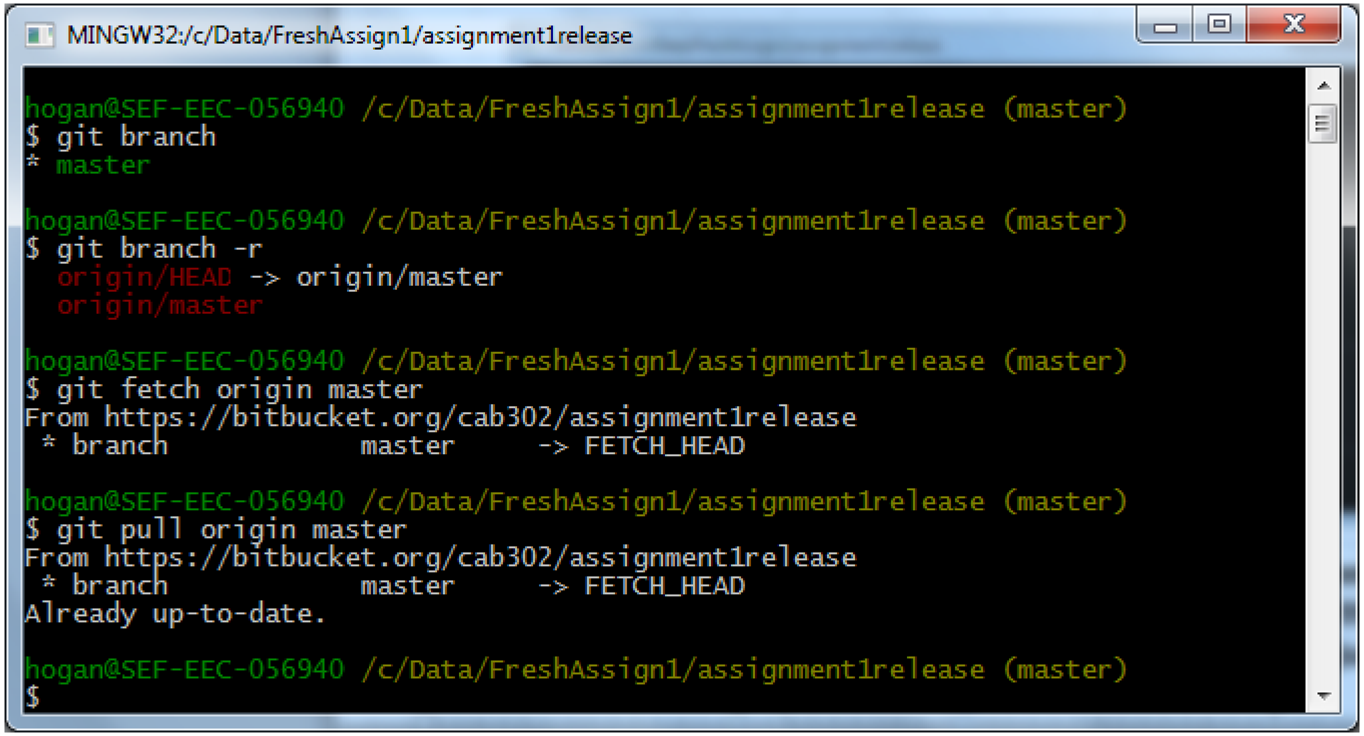
New to Bitbucket?

Learn the basics of using Git and Mercurial by exploring the Bitbucket 101.

Working in a team?

Create a team account to consolidate your repos and organize your team's work

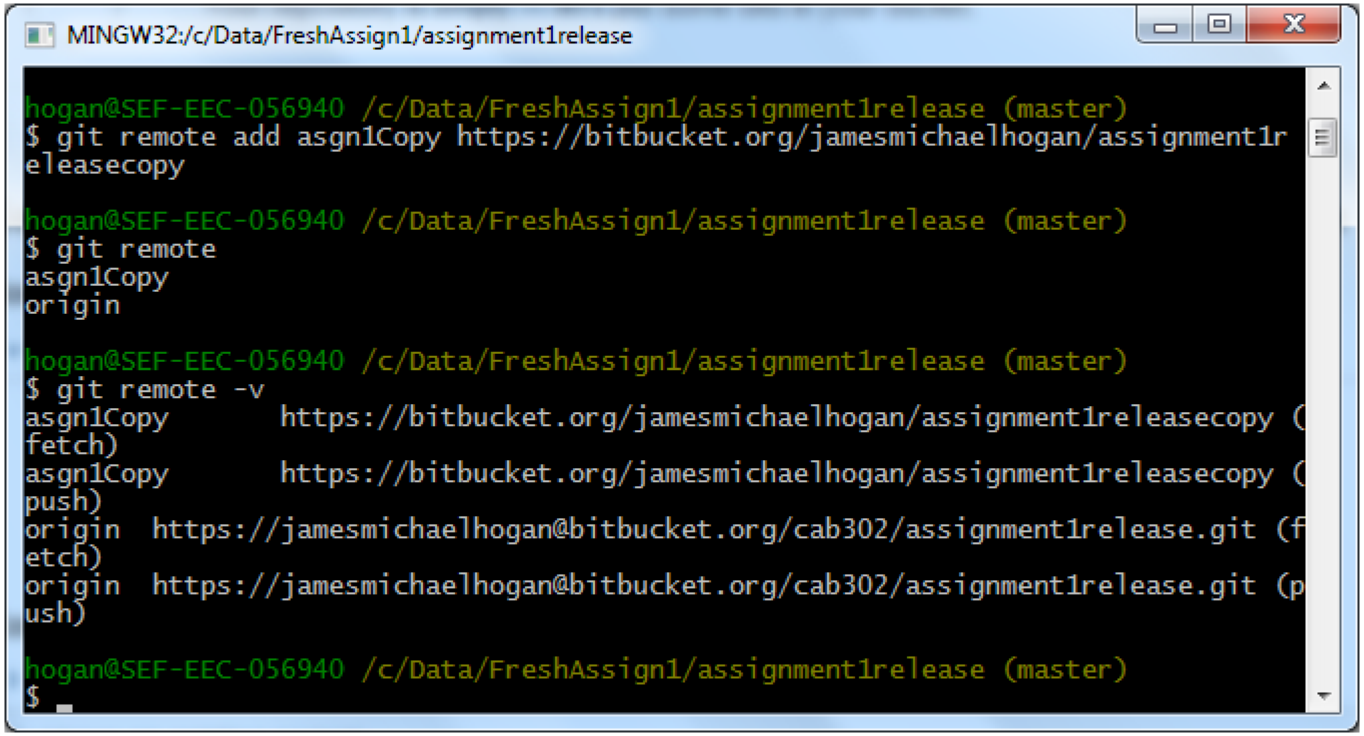
Remote and local branches


MINGW32:/c/Data/FreshAssign1/assignment1release

```
hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git branch
* master

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git branch -r
  origin/HEAD -> origin/master
  origin/master

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git fetch origin master
From https://bitbucket.org/cab302/assignment1release
 * branch            master     -> FETCH_HEAD

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git pull origin master
From https://bitbucket.org/cab302/assignment1release
 * branch            master     -> FETCH_HEAD
Already up-to-date.

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$
```

Here, branch shows the only local branch (master) and then with the option (-r) it shows the remote and the local cache of the remote. FETCH_HEAD is a temp pointer to what we have just fetched. Note that git pull does the fetch and the merge.

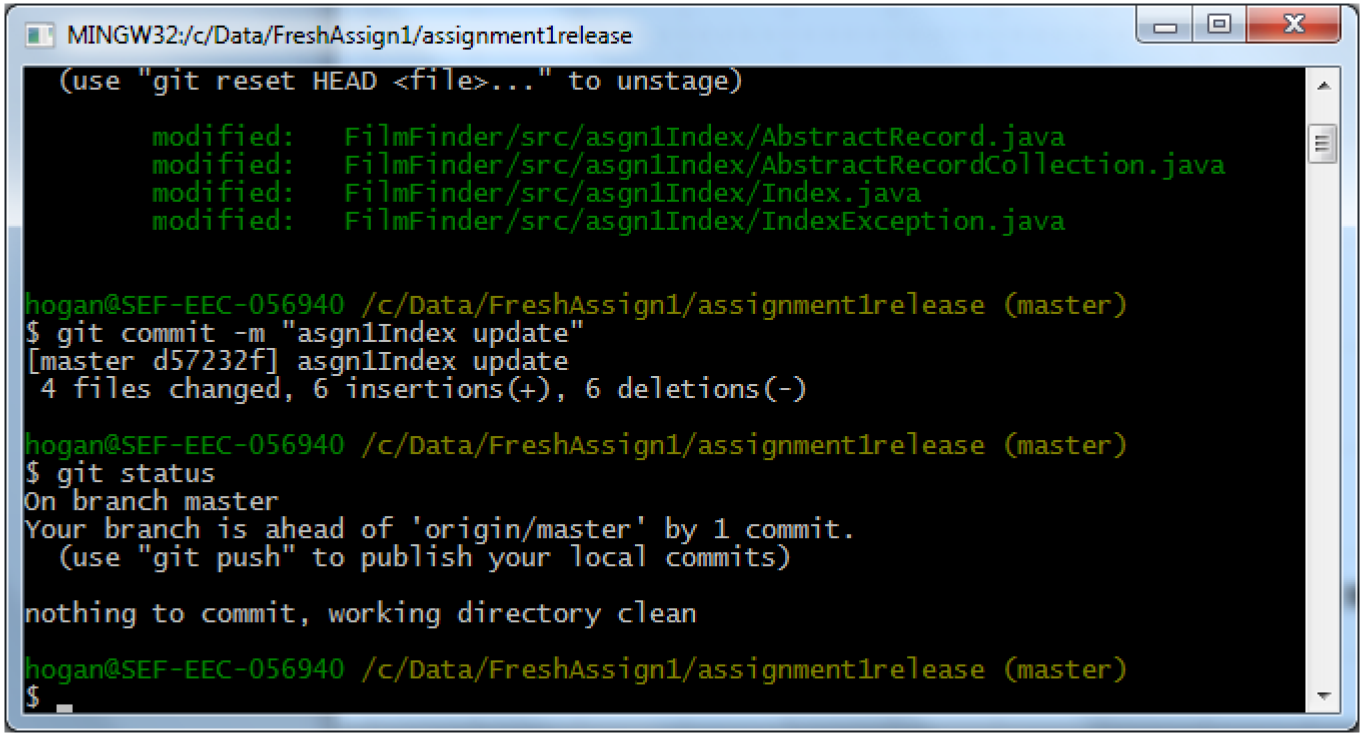
Another remote


MINGW32:/c/Data/FreshAssign1/assignment1release

```
hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git remote add asgn1Copy https://bitbucket.org/jamesmichaelhogan/assignment1r
eleasecopy

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git remote
asgn1Copy
origin

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git remote -v
asgn1Copy       https://bitbucket.org/jamesmichaelhogan/assignment1releasecopy (
fetch)
asgn1Copy       https://bitbucket.org/jamesmichaelhogan/assignment1releasecopy (
push)
origin  https://jamesmichaelhogan@bitbucket.org/cab302/assignment1release.git (f
etch)
origin  https://jamesmichaelhogan@bitbucket.org/cab302/assignment1release.git (p
ush)

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ _
```

Here, I add a new remote with the shortname asgn1Copy. Note that the remote URLs can be viewed by using the –v option.

Updated local repo

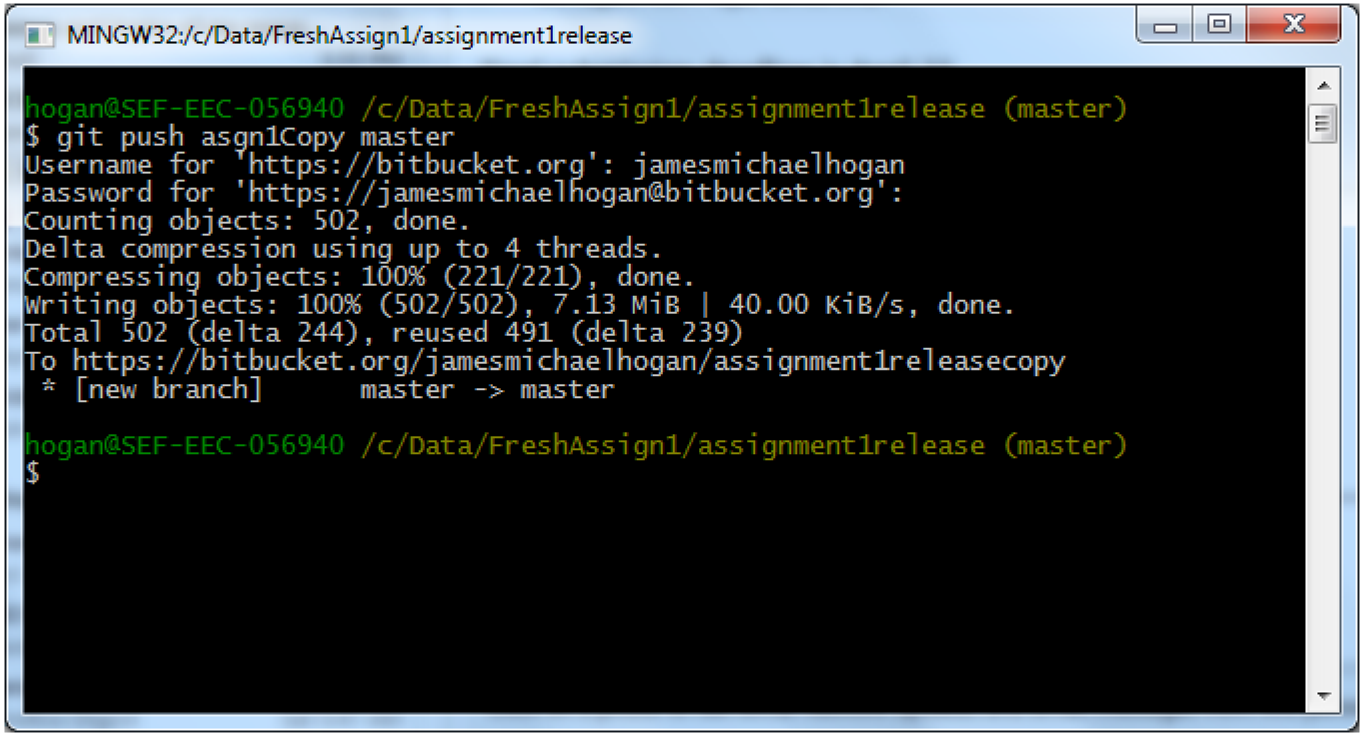


MINGW32:/c/Data/FreshAssign1/assignment1release

```
  (use "git reset HEAD <file>..." to unstage)

        modified:   FilmFinder/src/asgn1Index/AbstractRecord.java
        modified:   FilmFinder/src/asgn1Index/AbstractRecordCollection.java
        modified:   FilmFinder/src/asgn1Index/Index.java
        modified:   FilmFinder/src/asgn1Index/IndexException.java


hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git commit -m "asgn1Index update"
[master d57232f] asgn1Index update
 4 files changed, 6 insertions(+), 6 deletions(-)

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ _
```

Here, I have updated four java files and then committed them. We are now ahead of the origin by one commit.

Push to the remote


MINGW32:/c/Data/FreshAssign1/assignment1release

```
hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$ git push asgn1Copy master
Username for 'https://bitbucket.org': jamesmichaelhogan
Password for 'https://jamesmichaelhogan@bitbucket.org':
Counting objects: 502, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (221/221), done.
Writing objects: 100% (502/502), 7.13 MiB | 40.00 KiB/s, done.
Total 502 (delta 244), reused 491 (delta 239)
To https://bitbucket.org/jamesmichaelhogan/assignment1releasecopy
 * [new branch]      master -> master

hogan@SEF-EEC-056940 /c/Data/FreshAssign1/assignment1release (master)
$
```

Here, I push to the remote and it works – at least after I use the correct password.

Working with Git in a project

- Git integrates neatly with eclipse using the egit plugin.
- Installation follows the usual path (See prac)
- We can then operate by right clicking on the project menu
- Bring the source under Git control
- Usual operations are then available

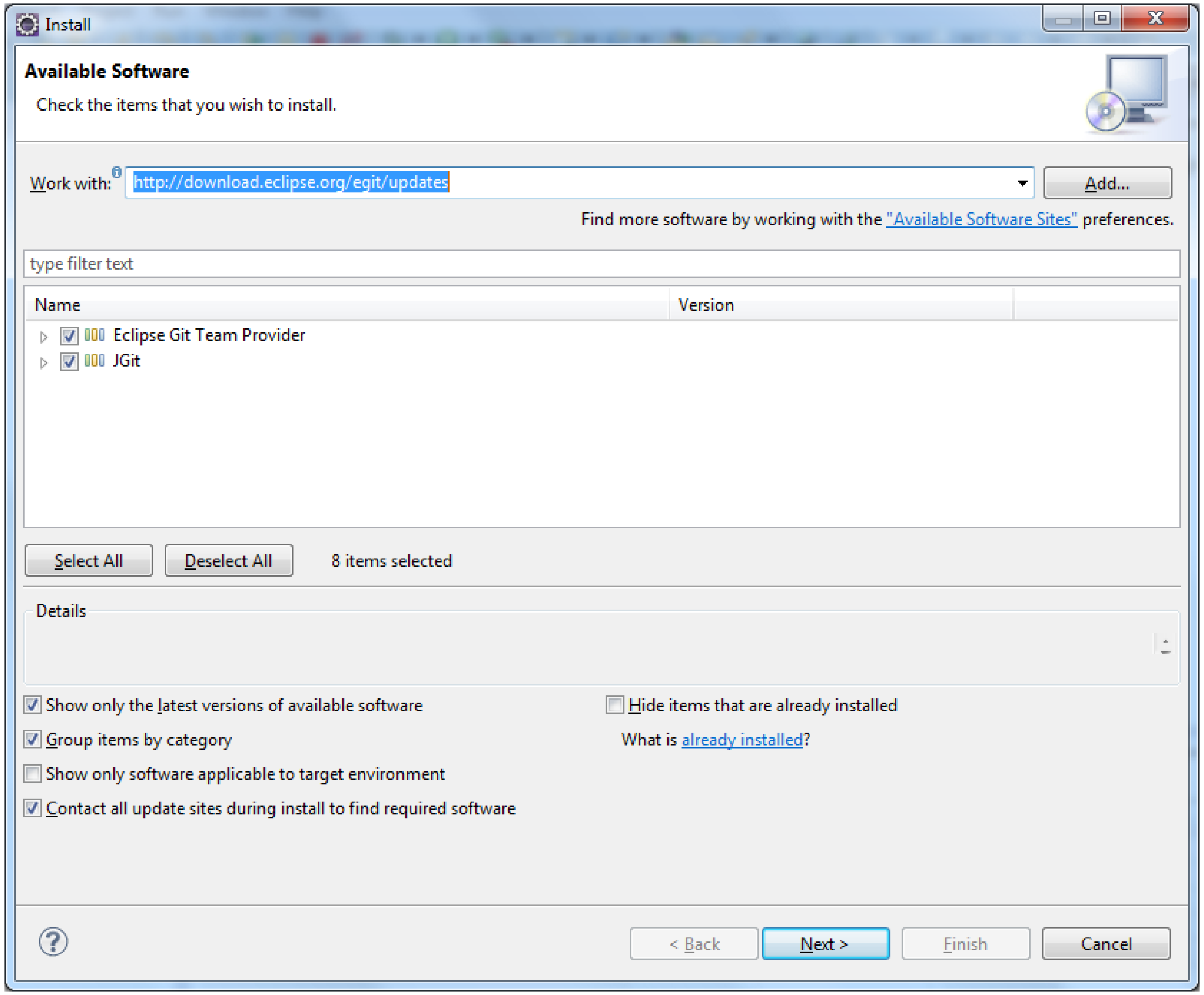
- External hosting is usually via one of two players:
- https://github.com/
- https://bitbucket.org/

- I am more familiar with bitbucket, so this is my preference, but this is not mandated… But see:
- https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101

GIT Tool Support

- You probably want to work with egit as it is the eclipse interface
- Alternatively you may use:
  - http://git-scm.com/downloads/guis includes std git tools, tortoise and others
- Egit is now part of the standard eclipse release
- For older releases use the help>install new software menu (see screenshot on the next slide). You will need to use:
  - http://download.eclipse.org/egit/updates
- Egit's default instructions are geared for GitHub. To work with Atlassian Bitbucket, please see:
  - http://wangpidong.blogspot.com.es/2012/05/how-to-use-bitbucket-with-egit-in.html

Egit installation



Install

Available Software

Check the items that you wish to install.

Work with: http://download.eclipse.org/egit/updates

Find more software by working with the "Available Software Sites" preferences.

type filter text

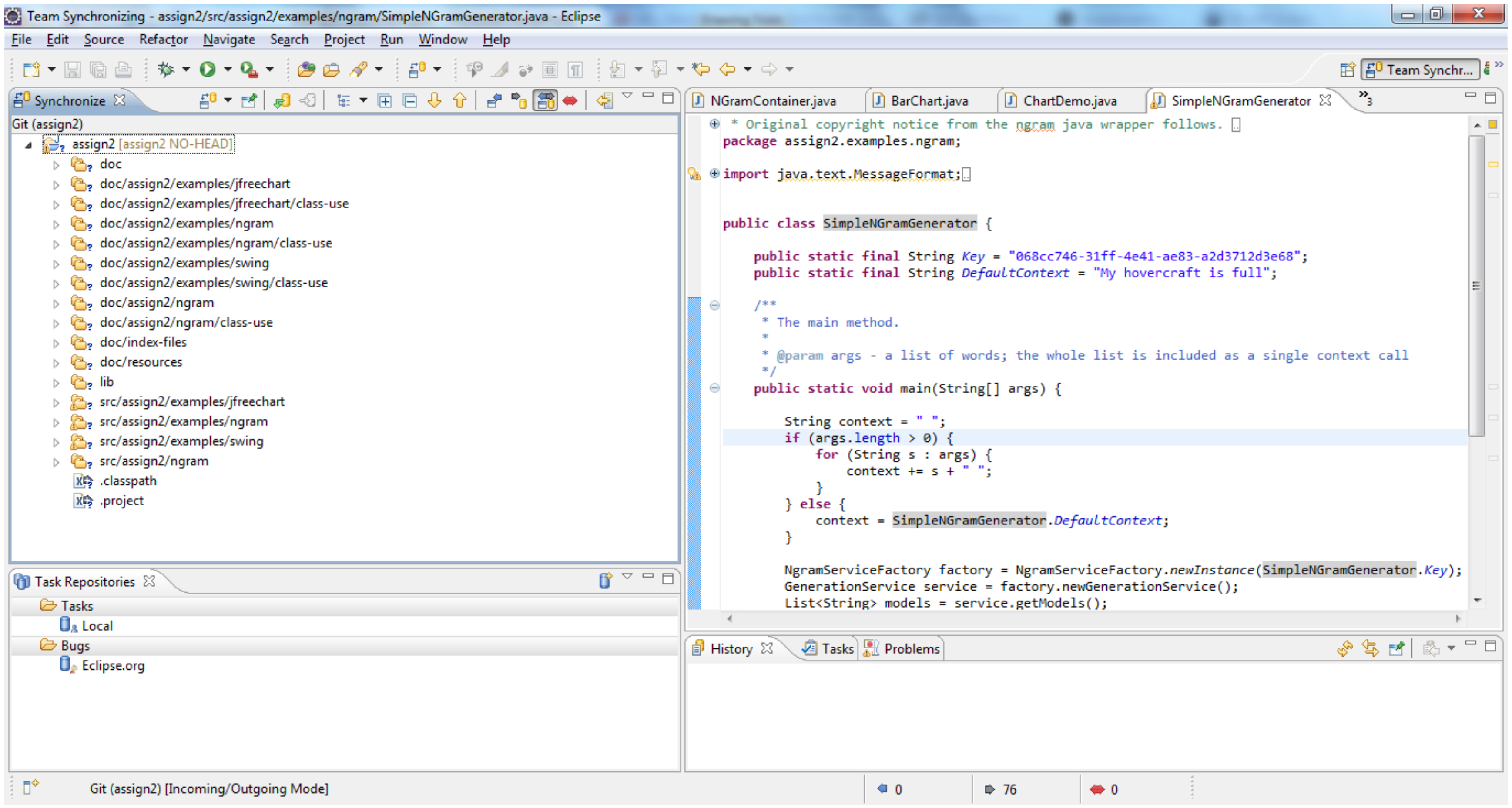
Name	Version
☑ Eclipse Git Team Provider	
☑ JGit	

8 items selected

Details

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☐ Hide items that are already installed
What is already installed?

egit


Team Synchronizing - assign2/src/assign2/examples/ngram/SimpleNGramGenerator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Team Synchr...
Synchronize
NGramContainer.java
BarChart.java
ChartDemo.java
SimpleNGramGenerator
Task Repositories
History
Tasks
Problems

```
Git (assign2)
  assign2 [assign2 NO-HEAD]
    doc
    doc/assign2/examples/jfreechart
    doc/assign2/examples/jfreechart/class-use
    doc/assign2/examples/ngram
    doc/assign2/examples/ngram/class-use
    doc/assign2/examples/swing
    doc/assign2/examples/swing/class-use
    doc/assign2/ngram
    doc/assign2/ngram/class-use
    doc/index-files
    doc/resources
    lib
    src/assign2/examples/jfreechart
    src/assign2/examples/ngram
    src/assign2/examples/swing
    src/assign2/ngram
    .classpath
    .project
```

```
Tasks
  Local
Bugs
  Eclipse.org
```

```java
 * Original copyright notice from the ngram java wrapper follows. 
package assign2.examples.ngram;

import java.text.MessageFormat;


public class SimpleNGramGenerator {

    public static final String Key = "068cc746-31ff-4e41-ae83-a2d3712d3e68";
    public static final String DefaultContext = "My hovercraft is full";

    /**
     * The main method.
     *
     * @param args - a list of words; the whole list is included as a single context call
     */
    public static void main(String[] args) {

        String context = " ";
        if (args.length > 0) {
            for (String s : args) {
                context += s + " ";
            }
        } else {
            context = SimpleNGramGenerator.DefaultContext;
        }

        NgramServiceFactory factory = NgramServiceFactory.newInstance(SimpleNGramGenerator.Key);
        GenerationService service = factory.newGenerationService();
        List<String> models = service.getModels();
```

Git (assign2) [Incoming/Outgoing Mode]   0   76   0

Aside: The World has Changed

- We had previously worked in this unit with SVN, as when we started out it was the modern alternative to the by then ancient CVS
- SVN and other ‘canonical’ repo source control systems have largely made way in the developer community for distributed approaches
- We have spent a good deal of time on Git
- But you should also be aware of Mercurial:
  - http://mercurial.selenic.com/

Both will continue

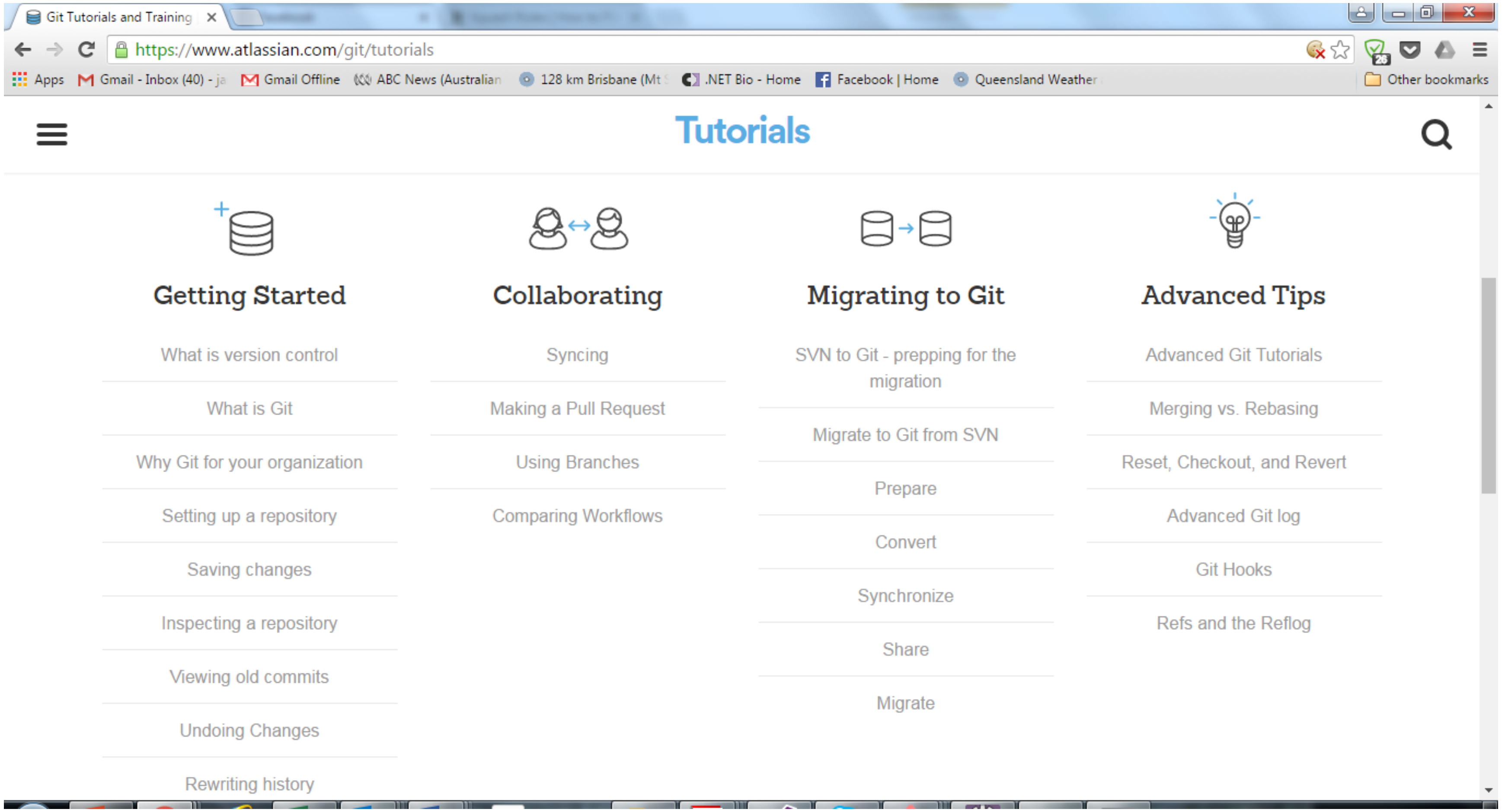
- SVN and its equivalents have certainly lost ground, but there remains some debate over the best choices
- Some religious wars, but mainly horses for courses
  - http://stackoverflow.com/questions/871/why-is-git-better-than-subversion
  - http://stackoverflow.com/questions/161541/should-i-use-svn-or-git
  - http://blog.marcomonteiro.net/post/30382607875/git-vs-svn-is-it-really-a-battle

- As noted, we have largely decided to move to Git, but we teach elements of both
- Don’t be too prescriptive or obnoxious in your views of other people’s source control choices
- It depends on context and project inertia

Version control summary

- The Version Control System helps us to monitor the state of the repository, to determine what changes have been made, and to manage concurrent changes
- As the system evolves, these changes can be automatically detected, and used as part of a fully integrated test and deployment environment
  - A common approach is to use Ant or Maven to specify *how* to build something
  - And then to use CruiseControl or Hudson to control *when* to perform builds

Tutorials


Git Tutorials and Training

https://www.atlassian.com/git/tutorials

Apps | Gmail - Inbox (40) - ja | Gmail Offline | ABC News (Australian | 128 km Brisbane (Mt | .NET Bio - Home | Facebook | Home | Queensland Weather | Other bookmarks

Tutorials

Getting Started

- What is version control
- What is Git
- Why Git for your organization
- Setting up a repository
- Saving changes
- Inspecting a repository
- Viewing old commits
- Undoing Changes
- Rewriting history

Collaborating

- Syncing
- Making a Pull Request
- Using Branches
- Comparing Workflows

Migrating to Git

- SVN to Git - prepping for the migration
- Migrate to Git from SVN
- Prepare
- Convert
- Synchronize
- Share
- Migrate

Advanced Tips

- Advanced Git Tutorials
- Merging vs. Rebasing
- Reset, Checkout, and Revert
- Advanced Git log
- Git Hooks
- Refs and the Reflog