Git as introduction to configuration management for students

Adam Nilsson and Marie Versland

2013-02-19

Contents

Abstract

In this report we have outlined the popular software configuration management tool Git. We have also touched on the associated software EGit, a Git plug-in for Eclipse and Github which is a hosting site for Git projects. The report gives a brief introduction to the basics of Git as well as descriptions of EGit and Github. We have had the opportunity to coach a group of 8 students during spring 2013 in agile software development. We used this occasion to test Git in practice in a somewhat small project, with the goal of finding out Git's usability and learning curve. We wanted to find out whether or not Git is a good alternative to SVN as a first encounter with software configuration management tools. The students behaviors and opinions have been recorded and analyzed in this report. We have used both literature and the student project to draw conclusions about Gits usefulness and its apparent success in the configuration management business. A comparison between Git and subversion is made, no introduction is made of subversion and basic knowledge is expected of the reader.

1 Introduction

We hear Git being mentioned more and more often and want to know why this is and what Git is all about. Software configuration management tools have existed for a long time and many different approaches have emerged. This report will answer questions about the viability of Git. The report is built on a course project performed at LTH.

First we will describe the three main components investigated Git, EGit and Github. We will go into more detail on Git since that is the central piece of our interest. We will explain Git's basic features that were essential to the workflow of our students, and the basic structure of the program. We will then outline the differences between Git and SVN and finally provide results from our study of the course project. The scope of this report is in relation to the project and therefore focus remained on the keyparts of Git and not on EGit and Github.

2 Git

Git is a distributed version control tool that was invented by Linus Torvalds to support the development of Linux. In April 2005 Git was used for the first time and Linus made the first commit with code for Linux. Linus named the version control tool Git which is an old English word for silly or worthless person after have claimed "I'm an egoistical bastard, and I name all my projects after myself" [1].

The purpose of Git is to provide a reliable and versatile version control and configuration management, it does this with a little different approach than some other version control tools. Git also enables people to work together and empowers teamwork.

REPOSITORY: Git is built around two parts your workspace and repositories. A repository in Git is similar to repositories or depots in other tools. The repository is where the files and information in the decentralized system is stored. From a repository you can extract information about the current version and earlier versions.Your repository act as the interface to other people you are working with as well as a powerful part in the configuration management process.

WORKSPACE: A workspace is where you do all your work. A workspace is private to one developer and can be said to be his or her personal area. When you edit a file, you do it in the workspace and no one else is affected, since it is your workspace.

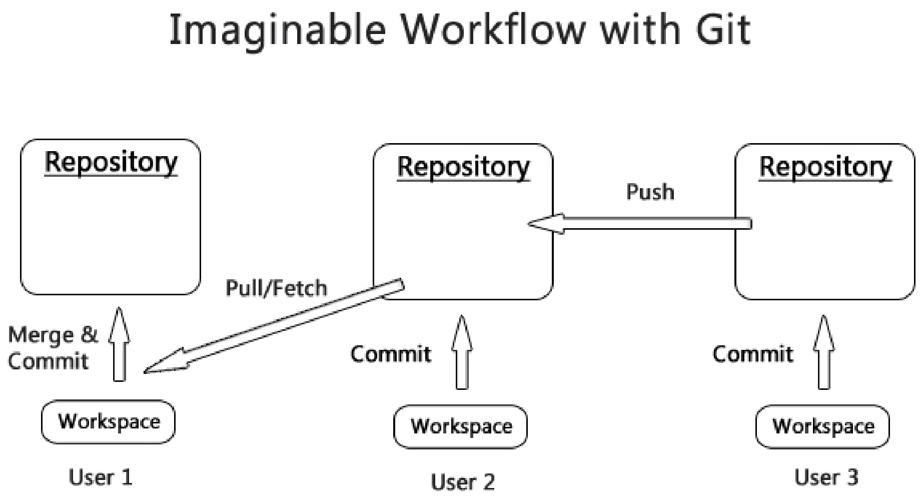


Figure 1: How data might be transferred within a system managed by Git

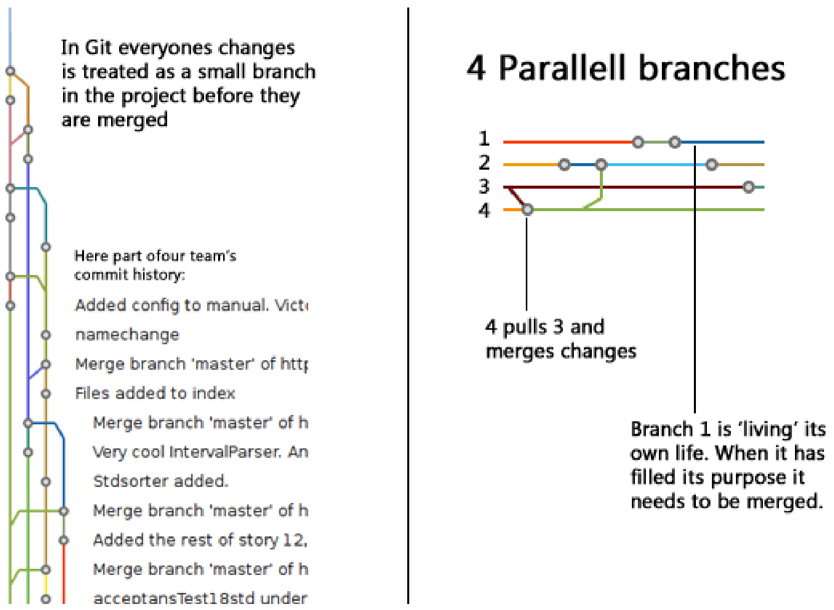
In Git along with your workspace you also get your own repository. When you have made a change you are content with, you want to save it. Saving it in the workspace would not accomplish that since this area of the system is changing all the time. You can make changes later that override the version you have now. This is one reason you want to have the repository, to save versions that you can later backtrack to if needed. Another reason is to efficiently share your work, since the repository can be made available to others to take from and even give to. Git saves a lot of useful information in the repository, it keeps track of who have pushed content to it, where and when merges in files has occurred and all older configurations of the system.

COMMIT/PULL/PUSH: When a piece of code is written the developer does a commit and the changes are saved in the private repository. Every commit can be identified by it SHA-1 hash that depends on its content [2]. By looking at two hash codes it is impossible to see which order the commits was made. An obstacle that needs to be solved by SCM-tools is when the user want to rename files. Git uses the content of the files to track and save changes, therefore it can easily discover if a file has been renamed rather than just tracking files by their file names.

When a developer wants some code that is in someone else's repository he/she can pull that into his/her own private repository. When a change is pulled, Git creates a new branch that is merged together with the local code and the developer gets all the data concerning that code e.g. the commit history. If there has been changes in the private repository since the last pull the merge are saved in the history as a own commit.

If a developer however wants to put its own code into some repository he/she can push it. But to be able to push the code the private repository must have pulled all the changes that exist in the targets repository.

BRANCHES: An important concept in software configuration management is branches, and it is even more prominent when using Git as opposed to other centralized SCM tools. In Git every commit in the repository has the potential to becom1 a branch [3]. The repositories is in a sense built around branches, things are stored in a branch [6]. To be useful, branches need to have things in common, if they don't they could just as well be and entirely different software project. Branches serve the purpose of isolating tasks during development to provide a stable base to work on without distractions. Branches can then be merged together resulting in a branch with the features of both of the initial branches. Often there is a 'main-branch' in a project which is what all small branches is merged to.


In Git everyones changes is treated as a small branch in the project before they are merged

Here part ofour team's commit history:

Added config to manual. Vict
namechange
Merge branch 'master' of http
Files added to index
Merge branch 'master' of h
Very cool IntervalParser. An
Stdsorter added.
Merge branch 'master' of h
Added the rest of story 12,
Merge branch 'master' of h
acceptansTest18std under

4 Parallell branches

4 pulls 3 and merges changes

Branch 1 is 'living' its own life. When it has filled its purpose it needs to be merged.

Figure 2: Example on branches

Git is based on the change set model [5], which means that changes to the software are stored as 'separate' modifications on a base configuration. This means that the changes can be applied independently of each other. This is of course only half of the truth since the functionality of the changes can depend on other modifications to the program. This is a powerful way of doing things but it also demands competence of the developer since faulty 'cherrypicking' of changes will result in buggy code.

Git also supports the concept of strict long transactions by default. This means that all code will always have existed together in a workspace before it can be pushed to a repository. This is achieved by making it impossible to push to a repository without being up to date with all its changes first.

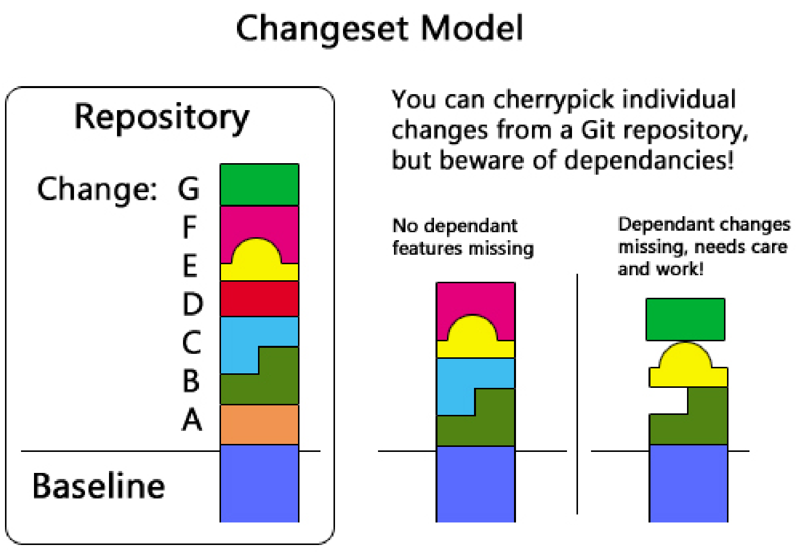

Changeset Model
Repository
Change: G
F
E
D
C
B
A
Baseline
You can cherrypick individual changes from a Git repository, but beware of dependancies!
No dependant features missing
Dependant changes missing, needs care and work!

Figure 3: The effect of Git using the change set model

One thing you want to do often when working on software as part of a team is integrating your co-workers work into yours. As Git uses private repositories you get a lot of freedom. When downloading other changes you do not have to merge them with your own changes immediately [3], this is because the private repository can store them seperatly in a branch so you can review the changes before deciding to merge. If you want you can even continue to work on your friend's changes in parallel with working on your own utilizing branches.

3 EGit

EGit is a plug in to the very popular Java IDE eclipse. EGit is a straightforward extension of the Git features into the eclipse interface. Instead of a separate client or text based console you get your commands from drop-down menus on the project in Eclipse.

When using a plug-in such as this you bring your version management tasks very close your programming activities. To entwine your development and your configuration management may or may not be a good idea. A plug-in can possibly decrease the awareness of the distinction between the two slightly disjoint activities of implementation and configuration management. We noticed this with our students when they argued over Egit and terminal.

4 Github

Github is a Git hosting service where a person can make his own repository accessible for the public. Github is free unless you want your code to be private. It also offers a lot of socially focused features [4]. Github supports Git and makes it more accessible to the public. There are a lot of code hosting services on the internet for all kinds of SCM tools if you do not want to trouble yourself with setting up servers and configuring the tools.

Github is very focused on the collaboration between users. Each project on Github get its own wiki-page where people active in the project can document various information. Github also tracks commits and makes statistics accessible via graphs. As a user you can follow other users and enter projects [4]. This is surely part of why Github is so successful, its features gives clear feedback on projects progress and allows for easy communication between developers. Many projects on Github are only used to store code and provide backup for single person projects [4].

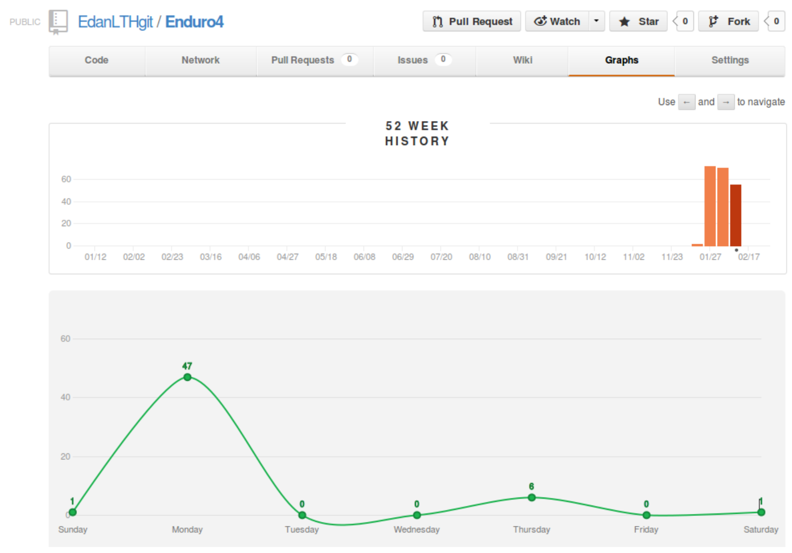


Figure 4: A graph produced by Github showing how code only gets committed on mondays in the course project.

5 Git versus SVN

After SVN Git is the most used version control tool in some areas [2]. Git is a decentralized software configuration management (DSCM) where every developer has his own repository. In DSCM work can flow sideways and between developers. SVN is a centralized SCM tool where work always flows up and down to a central repository.

Everyone in Git has his own repository, this makes it possible to share code without the use of a master repository, this is one of the biggest differences between SVN and Git. In SVN you are working in your workspace with one connected repository whilst in Git you are having your own repository as well as your co-worker's to keep track of, and to integrate changes you want continuously. In Git you have the freedom to take in half finished changes from specific co-workers to align your work, this can be very powerful but it also requires great awareness within the project.

Even if developers doesn't need a master repository like you have in SVN, in Git you can fake one repository to be the official one, where they can make releases and which code counts as the current main source code for the project. When using SVN, a commit is directly visible for all developers in the master repository but that is not the case in Git. Because in Git you are using private repositories a commit is only shown in that repository. The other developers doesn't see the commit before the have pulled in the changes to their own repository or the change was pushed into another repository. When using Git with a master repository you are actually creating a centralized working process. This way of working is not what Git was developed for and is limiting Git's potential, it still retains some interesting differences to SVN but much freedom is removed. That being said using a master repository can be a good way to start with Git if you are used to SVN [3].

Both SVN and Git has its own plug-in for Eclipse, for SVN the name is subeclipse. Plugins to famous IDEs such as eclipse can be important to spread a tool and for many people it could be a deal breaker when choosing SCM tool. The power of these two plug-ins are roughly the same in the sense of their relevance to their respective tool. They both seamlessly handle the Git/SVN tasks such as merging an committing. The Git way of doing things is a bit more complex and this reflects in EGit, providing more functions and more alternatives. This can make the tool more difficult to get into but for advanced users it provides all the necessary features of Git.

6 Course Project

The following sections describe the project we used as a base for this study and Gits performance within said project. The second section documents what happened in the project related to Git and students interactions with Git and their thoughts on the software.

6.1 Course Project description

The compulsory course eda260, software development in teams, is held every year for students in the second year at the computer science program at LTH. The students is given a project where they are to develop a system for registering times at an enduro competition. The course begins with a theoretical part and is then followed by a practical project part where the students are to develop the system. In the theory part the students learns about XP and SCM. In the second part they are divided into teams of eight to ten and expected to apply what they have learnt on the project. The project has six iterations and three releases, the system is developed during six 8 hour programming labs and six two hour planning meetings and every one of them must always be present.

We have helped the team by coaching them in the agile thinking and programming. Every team has gotten two coaches from the course eda270, coaching programming teams.

6.2 Git in the Course Project

We as coaches did have very limited previous experience with Git, EGit and Github. As amongst the focus of this study was to find what makes Git successful and what differ from SVN we found the project and the data it gave us very useful.

6.2.1 Git limitations

In the course project we did not use all the features that was available in Git which is because because of the small scope of the project, but also partly because of our inexperience. One of the great things with Git is that the developers can pull changes between them unfortunatly we did not take advantage of that. instead we had one repository where everyone could push and pull changes, like a master repository. This was a conscious choice by us, the work-model with a master repository is more straight forward and we felt it would be more safe to use that. We are very aware this takes a lot from what makes Git be Git and have taken this into account when analyzing the students opinions and behaviors.

Most of the students hadn't worked with Git before and they only used basic functions. However some of the students were more used to Git from before and they made use of more of the Git functions, like branching. The fact that some of the students had used Git before was great because they could help the others to solve problems that occurred.

6.2.2 Egit

Most of the students were using EGit and those who learned it became used to it and appreciated it. Even if some of the students had personal grudges against EGit they all liked Git. One of the complaints was that there was a lot of buttons that didn't do anything but this is most likely a result of lacking knowledge of the buttons in question. In our team there was one student that refused to use EGit and used the console instead. He did this with the motivation that he thought Egit was 'messy' and he already knew the console quite well already. It can be the simple resistance to learn new things when you know something that works that create this behavior. We see it as a small failure for EGit since its job is to be easy to use, accessible and provide all the features of the console.

6.2.3 Github

We used Github to host our project. We did this because we thought it could be very useful for the students to learn since it is easily accessible, popular and also because we wanted to learn more about why Git is so successful. Very soon when we started the coaching it became clear that with the limited time with students we had to cut our focus from Github. In the project the students have not come in any significant contact with any of the characterizing features of Github. Due to the course format 'following' fellow students and other similar features has been quite unnecessary. The Github statistics have been nice to look at but have not really provided any significant extra value. No meaningful conclusions about Github have been made because of the limited scope of the course. However to the extent we used it it looked promising.

6.2.4 Problems

During the labs we studied how the students behaved and what problems they had. During the first lab the had a lots of problem that derived from how Git is used and most of all how to handle merge conflicts. This is logical because most students have not experienced very many merge conflicts before this course and have to learn how to solve it, especially when they happen to commit a class file which always conflicts. During the next two labs it was much smoother and we couldn't see many problems related to merging anymore and they seemed to like working with Git which also can been seen from the result from the survey. On the contrary the has coaches from other teams that uses SVN tell us that theirs teams are so tired of SVN that they consider to change to Git, in the middle of the project.

That being said the following programming sessions was not completely without Git related problems. Problems such as adding files to specific ignore lists caused some 'headache' and certainly stole a fair bit of time. Another problem was that pulling just stopped working at times. These problems is most likely due to inexperienced use of Git and maybe a misclick here and there causing problems.

6.2.5 Survey

After the students had been using Git during three lab sessions we asked them to fill in an survey about Git. We constructed the survey with mainly first-impressions in mind as well as some background checks on previous experience within the team. Most of the questions where simple mutli-choice questions, but there was also some questions with text boxes where they could fill in their opinions.

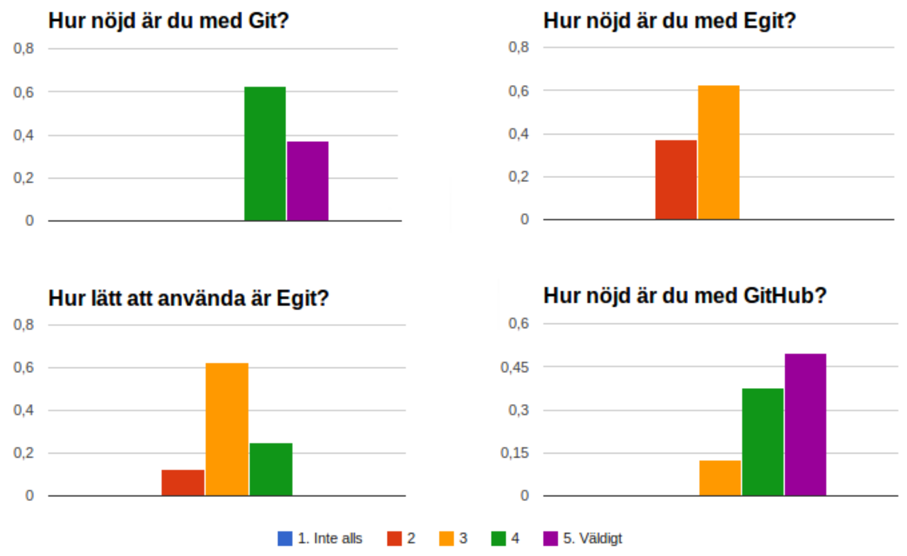


Figure 5: Part of the survey results

Something that the team liked about Git was that it was simple to use and that it handles branches better than e.g. SVN, which is in line with [6]. They also appreciated the private repository as a way to store local history and recognized the advantage it would give when working offline for some time.

7 Conclusions

Git is a powerful tool that can be really helpful while developing software. After having used it for a couple of weeks we can understand why its popularity is growing. It has many useful functions and it is easy to create and merge branches.

7.1 Easy to learn

Our group received limited training before the first session with Git. We had provided them with a document of simple instructions concerning the features they were expected to use right from the start in the project, See Appendix A. The study of our student group's behavior does not fully reflect Git because we used the master-repository setup. It would have been interesting to see how it would have worked if this was not the case.

When we started this project we didn't have any experience with Git or at least very limited experience, however we had used SVN though. In our team on the other hand most of the students did not have any significant experience of SCM nor any tools for it. When we started to look at and experiment with Git it took a couple of hours before we had gotten the hang of the most basic parts of Git like committing, pushing, pulling, fetching, branching and merging.

Many of the students in our team had a tendancy to have some problems with especially merging. But as we have stated above, after the first programming lab most of them got more control over merging and merge conflicts. We found that in EGit the way to merge conflicts are shown and resolved is a bit unintuitive. Having to use the "Add to index" button before committing to resolve merge conflicts is hard to figure out on your own, some of our students had either forgotten about this or not read our Git introduction carefully enough. Much like other SCM tools you need to work with the tool and abide by its rules rather than to try to force your old way of working to work with the tools

However it's not many of students that are using branching and some of them doesn't know exactly how it works, we feel this is fine as there was not that much room to branch in this project. Many other features that were not touched includes bisect, rebase and cherry picking amongst others. This certainly is not Git's fault but a product of the nature of the course project.

We have found that it's not that difficult to learn Git in its basic form especially if you have some experience of version control tools before. If you want to use Git to its full potential you need more time and first-hand experience. The more difficult thing to grasp is the configuration management process you need to employ when using Git. Using Git to collaborate with other people on the same piece of software is a great challenge.

7.2 Git suitable for a course project

When you are using a master repository in Git it is not that different from SVN in how you use it, especially if you only are using the basic features. If you do so the difference is that you have your own repository you must commit to before you can push it to the master repository. If you use Git like that for the courseproject it is not very difficult and you can still have some control over the code that is needed for the course. The extra version control in your own repository for your own use was appreciated by some of the students, but the value of it would certainly go up as the projects get bigger. The students did not express the need of using Git's full functiionality in terms of interacting with eachothers repositories without a master repository. When asked, they were sceptical of its use for them since they felt they had enough problems with pushing and pulling from the master repository.

Git solves merges very well despite the little hiccup in our team and it is not that often there becomes conflicts which is too difficult to solve. Our group did not have much troubles with the renaming and the deletion of files which we as coaches have had some troubles with in SVN before. The fact that it is easy to change the names on the files in Git is because it tracks the contents of the file makes Git more able than SVN where it can be troublesome to change the name of a file or removing a file.

All this and the fact that Git is easy to learn makes that Git is very suitable for the course project and all the students in our team prefer Git over SVN, one the other hand so have the never really worked whit SVN. But as touched on above in this document students of other teams is tired of SVN and wants to change to Git instead. EGit on the other hand did receive some negativity in the survey but many still prefer it over the console. The impression we have got when coaching is that it can be quite hard to solve some of the more 'mysterious' problems with EGit, and that it was lucky that one of our students knew the Git by console quite well. The 'mysterious' problems was the kind of problem such as certain functionality like pull giving errors with no explanation. The group only had these problems in the beginning and vanished in later iterations so we conclude they arose through misuse of the plugin.

Overall we think that Git is very usable as a base to learn SCM and XP from. Git both has advantages and disadvantages compared to SVN. Git gives more freedom to the cost of being more complex than SVN. The freedom given to the developers is not always desireable, in some cases a master repository provides a good place for authority to draw information from. Both tools are very adequate and we can recommend Git for the purposes of learning.

8 Summary

Git is a popular version control tool where everyone has his own repository and you can pull and push changes between the developers. If you make all exchanges of code go through one repository it becomes a master repository, you can work much in the same way as you do in centralized SCM like SVN. Git is quite easy to learn and and nice to work with and the students of course project seems to prefer it over SVN. Git have many powerful software relatives such as Github and EGit which is helping its popularity.

9 references

1. Version Control with Git by Jon Loeliger & Matthew McCollough
2. The Promises and Perils of Mining Git by Christian Bird, Peter C. Rigby , Earl T. Barr , David J. Hamilton, Daniel M. German, Prem Devanbu
3. Making Sense of Revision Control Systems By Bryan O'Sullivan
4. Social Coding in Github: Transparency and Collaboration in an Open Software Repository by Laura Dabbish, Colleen Stuart, Jason Tsay, Jim Herbsleb
5. Configuration Management Models in Commercial Environments, by Peter H. Feiler
6. Tech Talk: Linus Torvalds on Git. http://www.youtube.com/watch?v=4XpnKHJAok8. (Warning biased)

Appendix A

Installera Egit till eclipse
Help → Install new software → add

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

Välj *Eclipse Git Team Provider*. Om det inte fungerar gå tillbaka och klicka ur *EGit Plug-in Import Support* och försök igen.

Hämta hem projeket
File → Import → Git → Projects from Git →URI
Följ instruktionerna (klicka next) och använd nedanstående https och **välj att skapa projektet från existerande projekt** i sista steget.
https://github.com/EdanLTHgit/Enduro4.git

Nu har du ett project i ditt eclipse som är kopplat till ett eget privat repo och som har det stora master repot som ‘upstream’. Repositorierna på github är public så ni kan ta ner dem, men ni kommer inte kunna pusha saker till dem förrens ni är tillagda som collaborators.

Komandon i git
(Under Team fliken i context menu)

Commit spara dina ändringar till sitt eget repo.

Fetch hämtar från annat repo utan att merga ihop det med det man har i sitt, fetch används när man vill titta på det man hämtar ner innan man tar in det i sitt workspace.

Push lägger upp det man har commitat i sitt eget rep till vårt repo på GitHub, för detta krävs autentisering. Ex: Om du har commitat 2 gånger till ditt repo så kommer dessa båda commits läggas upp på github repot.

Merge lägger ihop sitt eget repo med det man har hämtat.

Pull genomför både fetch och Merge, detta är det ni kommer använda som ‘update’.

Vid fetch, push och pull ska följande https adress användas
https://github.com/EdanLTHgit/Enduro4.git
och vid push måste ni använda det användarnamn och lössenord till gitHub. Ni kan använda secure store för att spara lösenord så ni slipper skriva in det varje gång.

Show history - Visar en logg med alla commits och hur saker o ting mergats

Merge
När ni har fechat från ett annat repo så måste ni själva merga ihop det med er egen kod. Eftersom vi inte vill ha några konflikter i repot så måste fetch och merge eller pull alltid utföras innan push

När det uppstår konfliker vid en merge så måste ni lösa dem.
Team → Merge Tool
och rätta till, när filen är som den ska så lägga till filen genom
Team → Add to index
därefter gör en commit

(Man behöver inte använda merge toolen, man kan ändra i filen manualt också, det viktiga är att man tar ‘add to index’)

Viktigt!

Comitta aldrig .class filer till repot. Man ska kunna sätta dem på “ignore” under windows-preferences.

Experimentera

https://github.com/EdanLTHgit/LekRep.git