Collaboration Tools in Software Engineering

Stepan Bolotnikov
Me

- Stepan Bolotnikov
- Software Engineer at Proekspert
- MSc in Software Engineering from UT, 2018
- stepan.bolotnikov@proekspert.ee
You

Mostly BSc students in Computer Science

- Software developers / QA engineers
- CS researchers
- Project managers / team leads

In any case connected to software projects, sharing knowledge and resources
Course

- History and working principles of version control systems (VCS)
- Git distributed VCS
- Issue tracking
- Theoretical knowledge + practical hands-on exercises

- 7 sessions
- Every 2nd Friday
- Lecture + practice
- Non-differentiated (pass/fail)
Schedule

- 21. Feb - Introduction, history of VCS
- 06. Mar - Introduction to Git, setting up the first repository, basic Git usage
- 20. Mar - Common Git commands
- 03. Apr - Branching in Git, common branching models
- 17. Apr - Troubleshooting common Git issues
- 01. May - Video lecture; Mercurial
- 15. May - Advanced Git usage; git hooks and CI
- 29. May - Guest lecture, preparation for exam
- 01. June - Exam 1
- 05. June - Exam 2
Sessions

- 4h
- Lecture part
- Practical part
Final exam

- 1st or 5th June
- Individual practical tasks
- “Poor”, “Satisfactory” or “Good”
- “Satisfactory” and “Good” - passing
In order to pass the course

- Active participation in at least 5 out of 7 sessions
  - Complete the practical tasks
- “Satisfactory” or “Good” on the final exam
Communication

Course website

http://courses.cs.ut.ee/2020/cse/spring

Course Slack

Click
Lecture 1: Introduction to course, History of Version Control Systems
Today we will learn

- Course schedule, goals, topics
- What exactly is a version control system
- Why is VCS necessary
- Short overview of history of VCS
Version Control System

- Tracks changes in data
- Allows to view history of changes
- Allows to revert to previous version

Often part of an application (Google Docs version history; Wikipedia page history; Microsoft Office 365) or a stand-alone application (SVN, Git, Mercurial)

A set of changes or “revision” commonly identified by a code, timestamp and the person making the changes.
What if you’re not alone?
● Several developers, working on a project at the same time
● Project is periodically deployed e.g. to a web server
● Developer A needs to add his code
● How to know where is the latest version?
Rubber duck VCS (RDVCS)

- Several developers, working on a project at the same time
- Project is periodically deployed e.g. to a web server
- Developer A needs to add his code
- How to know where is the latest version?
- (or any other object, really)
How to use RDVCS

1. Find the developer with the rubber duck
2. Retrieve the rubber duck & latest version of code
3. Add your changes
4. Deploy the project
In a way, that is a version control system.

BUT

- What if the project is longer than a week or two?
- What if you have collaborators outside of office?
- What if you encounter a confusing piece of code?
- How do you go back to a certain state?
VCS software

- Automates some or all of the process
- Gives better overview of data
- Makes data manipulation simpler
- (Not necessarily just for source code; can be used with text; configuration files; images, etc)
But first, some terms

- **Repository** - where the history of file(s) is stored
- **Check in** - “upload” to the repository
- **Check out** - “download” from the repository
- **Working copy** - current checked out copy of the data
- **Revision** - a set of changes to the file(s) that VCS tracks
Repository as a graph

- Initial version is the root node
- Every revision is a child node of a revision before it
- Some VCS allow for “branches”
- Changes in branches are usually “merged” into the main branch (“trunk”)
Step 1: Local VCS
Step 1: Local VCS

- Earliest VCS (SCCS - 1972; RCS - 1982)
- Only on one machine
- Only on one file
- Only one user can access at a time
- Versions are stored as “deltas” - lists of changes
- Working copy - current checked out writable file
- Repository - history file
Source Code Control System (SCCS)

- Developed in 1972 in Bell Labs for IBM System/360 Operating System
- Later rewritten in C for UNIX
- First publicly released version - 1977
- Dominant VCS until RCS took over
- Other than fixing some Y2K bugs, has not been in active development
- Considered obsolete
How to use SCCS

Step 1: Create a new history file

$ sccs create program.c
program.c:
1.1
87 lines

- Creates a new history file (s.program.c) in SCCS subdirectory
- Retrieves a read-only version of the file
How to use SCCS

Step 2: check out an editable version

$ sccs edit program.c
1.1
new delta 1.2
87 lines

Retrieves a writable copy; locks the history file so nobody else can check in changes.

Step 3: write some code
How to use SCCS

Step 4: check in your changes

$ sccs delta program.c

comments? corrected typo in widget()
1.2
5 inserted
3 deleted
84 unchanged

Adds your changes to the history file with the given comment. Releases the lock.

Alternatively delget (delta + get) can be used to continue editing file after check in.
How to use SCCS

Step “something went wrong”: retrieve a previous version

$ sccs get -r1.1 program.c
1.1
87

$ sccs get -c880722120000 program.c
1.2
86

Retrieve a version by ID or retrieve a version that was latest at a given time.
Step 2: Centralized VCS
Step 2: Centralized VCS

- History is kept in a central repository
- Users check out latest state to their own machines
- Can handle several files
- Handles several users better
- Working copy - currently checked out project on local machine
- Repository - central repository kept on a remote server machine
Concurrent Version System (CVS)

- 1990
- Server
  - Has full project history
- Client
  - Gets files from server, sends new revisions to server
- Doesn’t track directories and filename changes - no renaming/moving
Apache Subversion (SVN)

- 2000
- Direct successor to CVS
- Handles renaming, moving, copying children.
- Became very popular in open-source community (e.g. SourceForge)
- Still developed and in use
How to use SVN

Step 1: Check out a repository

$ svn checkout path/to/svn/repository
A repository/base-file.txt
Checked out revision 1.

Downloads the latest state of the project on your machine.
How to use SVN

Step 2: Write some code (e.g. change base-file.txt and add test-file.txt)

Step 3: Check the state

$ svn status
M   base-file.txt
?   test-file.txt

? shows that the file is not under version control
M shows that the file has been modified
How to use SVN

Step 4: add the new file to version control

$ 

svn add test-file.txt

A test-file.txt

Step 5: commit your changes to the repository

$svn commit

Sending base-file.txt

Adding test-file.txt

Transmitting file data .

Committed revision 2.

Sends all the changes to remote repository
How to use SVN

Step 6: Fetch latest version from the repository

$ svn update
U test-file.txt
Updated to revision 3.

Downloads the latest state.

U shows that the file was updated
But what if your repository is lost?
Step 3: Distributed VCS
Step 3: Distributed VCS

- Instead of the latest version, clients check out complete history of the project
- Can work with multiple “source” repositories
- Designed to work with multiple users across multiple files
- Working copy - currently checked out project
- Repository - latest version of project kept on the local machine
- Repository is local, but can be updated from a different remote repository
Why distribute?

- Project can be restored from any “client” repository
- Developers can continue work offline
- Multiple “remote” repositories allow to collaborate in different ways with different people within one project
- Common commands are fast, because don’t require a remote server
BitKeeper

- Distributed VCS
- Originally proprietary software (later released as open-source)
- Free version used to develop Linux kernel 2002-2005
- 2005: support for free version ended, spurring creation of Git and Mercurial
Git

- Linux kernel could no longer be developed on BitKeeper
- Other solutions didn’t have sufficient features or were not fast enough
  - Performance - patching should take no more than 3 seconds
  - Take VCS as an example of what NOT to do
  - Support distributed workflow a la BitKeeper
  - Safeguard against corruption
- Linus Torvalds creates Git in 2005
What’s in a name?

Readme file of Git source:

The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your way):

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- "goddamn idiotic truckload of shit": when it breaks
Characteristics

● Non-linear
  ○ Several branches (working copies) can exist

● Distributed

● Fast
  ○ Order of magnitude faster than many VCS

● Cryptographic authentication
  ○ Snapshot ID is a SHA-1 hash of its contents and depends on history up to that point; impossible to change previous versions without being noticed
Snapshots, not differences

Most other VCS store information as a list of file-based changes (deltas)
Snapshots, not differences

Git works with snapshots of its own miniature filesystem. Every commit takes a snapshot of what the entire project looks like. Unchanged files are handled as links to previous version. Git can be seen as a filesystem with powerful tools built on top of it, not just a VCS.
Staging area

- Other VCS have two states of a file: modified and checked in
- Git has an additional state: staged
- Staged means that you have marked a modified file in its current version to go into your next commit
- Lets you choose specifically what to commit
Working Directory -> Staging Area -> .git directory (Repository)

- Checkout the project
- Stage Fixes
- Commit
Why git?

- It’s the best
Why git?

- It’s the best
- Fast
- Powerful branching capabilities
- Fine control over changeset
- Understandable syntax
- Open source
- Most popular VCS
- Used by the open source community as well as enterprise
How to use git

- Command line application
- GUI application
- Part of IDE
How to use git

- Command line application
- GUI application
- Part of IDE

- Deeper understanding of git design and commands
- Uniform terminology
- Better intuition and faster usage
How to use git

- **On Linux**
  - From Terminal (download from preferred package manager)

- **On Mac OS**
  - From Terminal (need to install it from git homepage or via homebrew (or from Xcode Command Line Tools))

- **On Windows**
  - Download git-bash from git-scm.com
  - Virtual Linux machine
  - Linux subsystem for Windows
  - From Command line
More on git & download

http://git-scm.com

https://en.wikipedia.org/wiki/Git
Practical part

Getting started:

1. Github
2. Installing Git
3. Making your first repository
References

- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dhp/index.html
- https://en.wikipedia.org/wiki/Revision_Control_System
- https://en.wikipedia.org/wiki/Apache_Subversion
- https://openoffice.apache.org/svn-basics.html