****

**Software Basics**

Lesson 3: Source control



**INTRODUCTION**

Every time when software is made there should be version reference. This include date and time of changes, purpose and person who did the change etc. At any given point the developer can roll back to the previous desired version.A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

A ***version control system*** allows programmers to keep track of every revision of all source code files. The main element of the version control system is the *repository*, a database or directory which contains each of the files contained in the system. A programmer can pick a point at any time in the history of the project and see exactly what those files looked like at the time. Changing a file will not unexpectedly overwrite any previous changes to that file; any change can be rolled back. There are two common models for version control systems.

* In a copy-modify-merge system, multiple people can work on a single file at a time. copy-modify-merge system- multiple people can work on a single file at a time. When a programmer wants to update the repository with his changes, he retrieves all changes which have occurred to the checked out files and reconciles any of them which conflict with changes he made before updating the repository.
* In a lock-modify-unlock system, only one person can work on any file at a time. In a lock-modify-unlock system, only one person can work on any file at a time. A programmer must check a file out of the repository before it can be modified. The system prevents anyone else from modifying any file until it is checked back in. On large projects, the team can run into delays because one programmer is often stuck waiting for a file to be available.

### Source control goals

The following are some of the goals of source controls.

* Manage the changes to documents, programs, files, etc.
* Track history
* Identify person responsible for each change, and reasons
* Recover (roll back to) previous versions as necessary
* Maintain sets of compatible versions of files (configurations)

Apart from the goals are the benefits associated with source control and these include:

* Branching and merging.
* Traceability of bugs
* history of every file.

**HISTORY OF EVERY FILE**

A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.

**BRANCHING AND MERGING**

Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

**TRACEABILITY**

Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

**INTEGRATED DEVELOPMENT ENVIRONMENT**

An **Integrated Development Environment** (IDE) enables programmers to consolidate the different aspects of writing a computer program. IDEs increase programmer productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging. IDEs are graphical in nature, meaning that they use windows and controls like buttons to display information and accept input from the user. For example, tools can include: text editor, project editor, tool bar and output viewer.

A **text editor**: a window for the input, arrangement, and commenting of programming language code.

A **project editor**: a window that lists all of the files that make up the software project.

A **tool bar**: a set of buttons that represent the functions the environment can perform.

An **output viewer**: a window that displays any messages that the environment generates during the operations it undertakes.

Most IDEs provide:

* **Syntax highlighting**- The IDE will highlight where programming language (syntax) has been violated.
* **Code completion** (through intellisense)- Makes use of features like intellisense which can give suggestions to the programmer once he or she types the keyword. **IntelliSense** is a code-completion aid that includes a number of features: List Members, Parameter Info, Quick Info, and Complete Word. These features help you to learn more about the code you're using, keep track of the parameters you're typing, and add calls to properties and methods with only a few keystrokes.
* Refactoring-*Refactoring* is a programming technique in which the design of the software is improved without changing its behavior.
* **Version control -** Keeps track of software changes.
* **Debugging**- IDEs are also used for debugging, using an integrated debugger, with support for setting breakpoints in the editor, visual rendering of steps, etc.
* **Code search**- IDEs can use different kinds of user interface for code search, for example form-based widgets and natural-language based interfaces.

Examples of popular IDEs include: Eclipse, Visual Studio Code, NetBeans, IntelliJ, IDEA, PyCharm, Atom, IDLE, Xcode, Komodo edit, BlueJ etc. Software developers have a choice on the use of IDEs depending on personal or organizational preferences.

**Installing Git**

Install Git on Windows

Git for Windows stand-alone installer

Download the latest [Git](https://git-for-windows.github.io/) for Windows installer. When you've successfully started the installer, you should see the **Git Setup** wizard screen.

Follow the **Next** and **Finish** prompts to complete the installation. The default options are pretty sensible for most users.

Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).

Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own.

These details will be associated with any commits that you create:

$ git config --global user.name "Emma Paris"

$ git config --global user.email "eparis@atlassian.com“

*Optional: Install the Git credential helper on Windows*

Bitbucket supports pushing and pulling over HTTP to your remote Git repositories on Bitbucket. Every time you interact with the remote repository, you must supply a username/password combination. You can store these credentials, instead of supplying the combination every time, with the Git Credential Manager for Windows.

