Getting Started with Source Control

# About Source Control

Your code is a valuable asset to yourself and your team. As soon as you start writing code, for applications, for automation, configuration as code, or analytics – you should start thinking about **Source Control.**

The term Source Control, and more generally Version Control Systems (VCS), refers to the tools and processes you use to:

* Store your code as a cohesive set of files
* Track changes to your code over time, across multiple new or deleted files
* Allow multiple copies of the codebase to exist at a point in time, using branching

In modern development practises, effective Source Control also becomes an integral part of Continuous Integration and automation, and allows you to implement approval mechanisms for accepting new changes.

Source Control is as relevant to existing projects as it is to new projects – although you will not have history prior to the implementation.

Source Control repositories can exist beyond the life of a project and act as a reference for you, your team or even the public (if you choose)

# Using Source Control

Source Control is meant to work for your team; whether that means a team of one, small teams, large teams, or even Open Source Software.

# Step Zero – Who are you sharing with?

Its worth considering before you start your project whether it is intended to be Open Source\*, or hosted privately. Even if it is not Open Source now, is there a chance that it might become Open Source one day? Different Source Control hosts have varying communities behind them and

*\* if Open source, what licensing do you plan to use – GPL, MIT, BSD etc. You may be restricted somewhat by the licensing model of component/libraries you use in the system – if in doubt, do more research or reach out to specialist licensing consultancies.*

# Step One – pick an approach

Step one is normally to decide on the Source Control approach. At the time of writing, Git has emerged as arguably the most popular technology to host code upon, largely because Git:

* Is free and Open Source in itself
* Is relatively easy to learn and understand
* Has a number of popular hosted implementations – e.g. GitHub, GitLab, Bitbucket
* Is a “Distributed VCS” and does not require a central version/trunk
* Works across OS platforms (even if your code doesn’t!)

Other Version Control Systems are available and better suited to certain teams/environments:-

* Microsoft Team Foundation Version Control (TFVC) now part of Azure DevOps Pipelines
* Subversion (SVN)
* Mercurial
* Concurrent Versions System (CVS)

# Step Two - Decide how you’re hosting Source Control

Although Git, TFVC, SVN or Mercurial are the underlying technology, how you’re hosting your repositories matters. For example, at the time of writing certain vendors have different licensing and/or pricing agreements for numbers of users, numbers of projects, number of private repositories. Most vendors support only a handful of Version Control Systems – normally Git is the primary choice.

Clearly, hosting your code with some of these software vendors has other advantages, for example integration with other products that the vendor offers. In the case of a decentralised VCS like Git you *could* opt not to use a hosted SAAS solution e.g. in defence work, healthcare… But the hosted option is the best fit for most common scenarios.

GitHub is arguably the most popular Git implementation and integrates neatly with most CI systems.

In the case of Bitbucket from Atlassian, the software integrates your repository neatly with Bamboo, Jira and HipChat.

# What do you use?

In the University of Exeter IT department we use Bitbucket. In addition to working with Jira, this is due to a combination of free education licensing and access private repositories. This means that the code we write is not accessible to people outside of the team, unless we invite them to contribute.

In addition, Bitbucket has an easy to use Desktop Application called SourceTree that makes it easy to use without resorting to command-line instructions. GitHub has a similar desktop application, GitHub Desktop, which also makes getting to grips with Source Control considerably less painful!

For personal projects I tend to favour GitHub due to its strong community, wealth of material and support and also its popularity in the developer world.

# Hang On, What’s a Repository?

A repository (or “repo” for short) is a very much like a folder – it is essentially a root folder for all the components relating to your work. Normally you will have one or more Repositories per Project, for example a repository for Application A, another for Application B, another for resources.

What to put into the same or different repositories is up to you and your team. You might want all your build scripts, config and documentation in the same repo as your application, meaning that updates are perfectly in sync throughout the project lifecycle. Also, bear in mind that the more things you put into a single repo the more things need to be synced each time you check for changes.

## How does it work?

Firstly, you create a repo.

Then, you are your team can **clone** the contents of the repo to your local laptop

You carry on your day job and change things in this folder – creating or updating code, config, docs, even deleting files

Once you reach a suitable state of change (and when all of the changes leave the folder in a working state!) you can **commit** your changes. It is good practise to *always* add a comment with each commit, including any suitable **tags**. One example might be the ticket number or Jira Story.

At this point, your commit is tracked on your local PC, like a waypoint. This preserves the order of commits.

Next, you can **push** your changes to the rest of the repository using the push command. This allows the remote repo and your team to see your changes.

Your team mates can then use the **pull** command to clone your changes down to their local folders.

At the point of using pull your Source Control software will throw an error if you have modified any of the files impacted by the incoming changes (since your last sync). This is called a **merge conflict** and requires you to manually resolve the situation safely before proceeding.

It is good practise to remember to **pull before your push**!

# Step 3 - Getting Started with Bitbucket

Create a remote repo

Pick a local folder

Clone down

Make changes

Commit

Pull

Push

Create a merge conflict…?

# Is there a quicker way?

Yes – no git tutorial would be complete without CLI Version… You might need to sign in via your browser!

# prep a folder

cd C:/Projects/Repos

mkdir Sandbox

# clone the repo (need to be signed in with Bitbucket in your browser)

git clone https://SimonStride@bitbucket.org/etlbi/sandbox.git

cd sandbox

ii .

# \*\*\*make changes\*\*\*

# review changes

git status

# use stage “.” to prep all changes

git stage .

# get the latest from everyone else

git pull

# commit to Source Control with a sensible message

git commit –m “my example commit message”

# sync your changes to the rest of the world

# change this bit to commit to a different branch

git push origin master

# Branching

Branching is a powerful concept that facilitates keeping multiple versions of your project going in parallel. For example,

* Release branch
* Test Branch
* Development Branch
* Individual Branches per developer

Again, how you use this feature is up to you and your team, and there is no “one size fits all” approach.

Code is moved between branches by **merging** commits from one branch into another.

# Tips

Do not EVER put passwords or sensitive data into a Source Control repository – even if private

Once you’ve setup a team repository, figure out what branch you want to treat as your production version. Often this is the “master” branch. Immediately create development branches to start working on.

Binary files – images, music, video, pdfs, don’t play particularly nicely. It can be done – but think twice about doing it. They take a long time to clone, pull or push. The pain also comes at sync and compare time - the files are either identical or they are not – there is no “line by line” comparison like in a text file. In addition, some hosts enforce a file size cap e.g. GitHub.

Avoid storing data of any kind in you version control system – it will slow everything down, and may not be relevant to all environments and all copies of the repo. You may also be impacted by the file size cap and if the information is PII remember your repository may be subject to GDPR legislation. You may also be stung if your repo ever goes public/Open Source.