**https://nhs.io/git/**

**Essential Git**

*Essential Git is a beginner’s tutorial to using Git on a Windows based machine. It covers the command line tools & how to use the Git work flows.*

**Introduction**

This Git introductory guide assumes no previous knowledge of using the Git source code control system, but progresses quickly towards an intermediate level of instruction … all the while sneaking in little nuggets of advice and best practises. It serves as a textbook, a guide for self-study, and a reference for Git commands and work flow scenarios.

While it's targeted at Visual Studio developers working on a Windows platform, all the commands and concepts should work on other platforms like Linux and Mac.

Towards the end of 2010 my employer started reviewing different source code control systems, and Git was chosen for it's speed, flexibility and distributed architecture.

Our team of six developers predominately works on the Windows environment, creating medical software using Visual Studio. We could have opted to use one of the many Git graphical front-ends, however we quickly realised that taking the time to learn and fully master the Git command line would make us more productive and give us a transferable skill that future employers would find useful.

As the person tasked with introducing Git into the organisation, I created this guide as an internal reference for our development team. It has since been rewritten, revised and published in the hope that other companies looking to incorporate Git will find the process painless.

The examples in this document have been tested against Git version 1.7.11 – the latest version of Git can be downloaded from the [Git web site](http://git-scm.com/).

**Getting started with Git**

**Installing Git**

To get the latest version of Git for Windows, visit the projects homepage on [git hub](http://msysgit.github.com/), find the downloads link and select the highest version number of the file, for example:

Git-1.8.1.2-preview20130201.exe

Download the exe file to your local hard drive, double mouse click the downloaded file to start the installation wizard and follow the on screen instructions.

**Starting Git**

Using Windows Explorer, browse to an empty folder you want to use as your working folder for a given project. Right mouse click, and select the option “Git Bash Here”. This will start a terminal window with Git running. To exit Git simply type exit at the command line.

paulx030a@AN111576 /c/$\_projects/Git

$ exit

**Exiting Git**

To exit Git, simply type exit at the command line and the DOS window will close.

$ exit

**Checking the version number**

To check the currently installed version number of Git, enter the version command and press return, and the full version number will be displayed on screen.

$ git version

git version 1.7.11.msysgit.1

**Getting help on commands**

You get get a list of common Git commands, by typing:

$ git help

To load the help file for a given command, simply append the command name to the above syntax. The help file will be launched into your default web browser for viewing / printing. So for example to view the help file for the branch command use:

$ git help branch

**Setting Username and Email**

Git keeps track of a developers contributions to a given project by saving the developers name and email address with each commit. These settings can either be set globally for every project the developer works on, or can be be changed on a project by project basis if required.

To set the values globally on all projects use the following commands:

$ git config --global user.name "Paul Bradley"

$ git config --global user.email git@paulbradley.org

If the developer then also wants to contribute to an open source project, outside of work, then he can set a different username and password for that Git repository by using the following commands:

$ git config user.name "k9directory"

$ git config user.email k9@paulbradley.org

Alternatively these settings can be applied by editing the relevant configuration files with a text editor. The values can be set or amended by adding or changing the following 3 lines in the relevant file.

[user]

name = Paul Bradley

email= git@paulbradley.org

The global settings can be found in the .gitconfig file which can be found under the documents and settings folder on C: drive. If your working in a corporate environment that uses Active Directory for logins, then the file will likely be in a sub-folder within the documents and settings folder which matches your profile name.

For settings local to a particular project, you need to find the config file within the .git folder under the projects folder.

**Setting up an .gitignore File**

By default, Git tracks content changes within all files under a given project tree. Depending on your programming language, there may be certain files that you don't want or need to track. For example temporary files produced by your compiler or application generated files like your error log file.

For Visual Studio projects there are certain sub folders and temporary build files that we don't want Git to track. You can tell Git which files to ignore by having a “.gitignore” file in your projects root folder. The contents of this file should contain a list of files or file types to ignore.

To create a new blank file in your project folder, use the following command:

$ touch .gitignore

Open the “.gitignore” file in a text editor, like Notepad, and enter the files or complete paths you wish to ignore. Below is an example of what should be ignored in a Visual Studio project:

/bin

/obj

\*.user

\*.suo

\*.sln.cache

Thumbs.db

Also add any application specific files you don't want to be tracked. For example I add my error log file.

**Creating a Bare Master Repository**

This function is only run once, at the start of the project, usually by the projects creator. To create a new blank repository for your project, use the following syntax:

$ git clone --bare my\_project my\_project.git

Substitute “my\_project” for the actual project name e.g.

$ git clone --bare alma alma.git

This will create a sub-folder under your project folder, called alma.git - you then need to move this folder onto the server where the master repository will reside.

**Create a Blank Local Repository & Commit Initial Project Files**

To make a blank local repository call the init command, and then add all the files in the project folder. Use the commit command with the dash m option to record the comment for the initial commit. This will set-up a master branch within your local repository.

$ git init

$ git add .

$ git commit -m 'add initial project'

**Create a Development Branch**

You never want to develop on the master branch or the development branch, but first of all lets make a development branch :-

$ git checkout -b development master

**Adding the Remote Master Repository**

Once the bare master repository has been copied to the intended server, then you need to add a reference to it in the local repository, so that you can perform fetch and push operations. The command syntax is shown below, where [name] is a short hand name for the project, and the full path to the master repository. As Git is based on Unix, the file path needs to have forward slashes instead of backslashes when used on Windows.

$ git remote add [name] [master git location]

An example for the ALMA project, would be

$ git remote add ALMA //Alma/Git/alma.git

Where Alma is the server name, Git is a sub-folder on that server, and alma.git is the master repository folder.

**Pushing Local Changes To The Remote Master Repository**

To copy your local master and development branch back to the server you need to use the push command. The --all switch tells Git to push both branches to the server. The server name is the shortname you used when adding the remote master repository.

$ git push --all ALMA

Which will output something like:

Counting objects: 26, done.

Delta compression using up to 2 threads.

Compressing objects: 100% (24/24), done.

Writing objects: 100% (26/26), 9.11 KiB, done.

Total 26 (delta 8), reused 0 (delta 0)

Unpacking objects: 100% (26/26), done.

To //lthtr-alma/Alma/Git/alma\_testing.git

\* [new branch] development -> development

\* [new branch] master -> master

Now if you use the branch command with the dash a switch, you can see both the local branches and the master repository branches, flagged as remotes. The current branch is flagged with the star (\*).

$ git branch -a

\* development

master

remotes/ALMA/development

remotes/ALMA/master

**Fetching The Master & Development Branches From The Remote Master Repository**

If you try to push your local changes to the remote master repository, which has since been updated by another user you will get an error message. In this case will need to fetch the remote master branches and merge them with your local copy, fixing any merge conflicts, before you can re-push your local branches to the remote master repository

$ git fetch ALMA

To merge the master branch:

$ git checkout master

$ git merge ALMA/master

... fix any merge conflicts ...

To merge the development branch:

$ git checkout development

$ git merge ALMA/development

... fix any merge conflicts ...

**Git workflows**

**Creating a Feature Branch**

**Must merge back into : development**

Branch name format : feature-\* e.g. feature-spell-checking

$ git checkout -b <featurename> development

Code the new feature, committing changes as you go:

$ git commit -a -m 'descriptive comment'

When your ready to check back into development, do:

$ git checkout development

$ git merge --no-ff <featurename>

$ git branch -d <featurename>

$ git push --all ALMA

**Creating a Release Branch**

**Must merge back into : development and master**

Branch name format : release-version-number e.g. release-1.6.0

$ git checkout -b release-1.6.0 development

Bump the version numbers

$ git commit -a -m 'bumped version 1.6.0'

When your ready to check back in, check into master first:

$ git checkout master

$ git merge --no-ff release-1.6.0

$ git tag -a 1.6.0 -m '1.6.0'

Then merge into development:

$ git checkout development

$ git merge --no-ff release-1.6.0

... fix merge conflicts ...

$ git branch -d release-1.6.0

**Git tips**

**Using TAB completion**

The Git command line supports tab completion, which will greatly increase your typing efficiency. Basically, you can type the first couple of characters of either a Git command or a branch name, press the TAB key, and if only one single entity can be found which matches those first couple of characters, then the system will auto fill the rest of the text into the command prompt.

So using the help command as an example, you could type the word “git” followed by the letter h, pressing the tab key will expand the letter h to help. As no other commands start with the letter h, the command line interpreter knows you must mean the help command. You can then insert a space and type br followed by the tab key to auto-expand br into the word branch.

This simple example has only saved you from having to type seven characters, however tab completion becomes very useful when dealing with branch names. It means you can have long descriptive branch names, without the fear of having to re-type the whole name when performing branch management tasks.

**Other useful commands / keyboard short cuts**

You can use the up and down arrow keys to cycle through the history of commands you have entered during the current Git session. This is useful if you want to review a series of commands, or if you make a typo within last command you entered. Instead of having to retype the whole command, you can use the up arrow key, to bring up the last command and then just edit the line correcting your error.

Git comes bundled with a lot of Linux commands, one of the more useful include the directory listing command “ls” – typing “ls” at the command prompt will display a list of all the file names in the current project folder. To get a complete directory listing, which includes filename, file size, date and time stamp as well as ownership, use:

$ ls -l

You can also search for specific files or file types. For example to search for all files with a vb extension use:

$ ls \*.vb -l

Which will output the file names in alphabetical order.

You can clear the command line window at any time by typing “clear” or by using the keyboard short cut CTRL+L.

Another useful command is the word count program, which can be used to calculate a word count of your documentation, by typing the following command:

wc -w filename.txt

Where filename.txt is the name of the file you want to process. Alternately, you can quickly count the number of lines of code your project by doing:-

wc -l \*.vb

**The integration work flow model**

Git supports several different work flow models and most corporate environments adopt the integration manager model – where a single person is responsible for all the commits to the “blessed” repository.

Developers clone from this blessed repository when needing to work on the project, pushing their changes back to their own independent repositories, asking the integrator to pull in their changes for approval and inclusion into the next release.

**Why pick Git**

Like many forward thinking employers, the hospital I work for realised the benefits of letting it's software development team work from home. Each developer has VPN access into the hospitals main server infrastructure, but the network access could often be slow which prevented us from using an always connected version control system.

The distributed nature of Git, allows our developers to work locally on their projects, making them more productive as they can work when and where they like without being restricted by having to be connected to a central server.

Once a developer has finished working on a new feature or bug fix, they can push their changes to the hospitals central Git server over the network, and because Git is so efficient at this process, even large changes don't take long to check in.

Another reason for choosing Git was it's flexibility around creating branches, and more importantly the ease at which these changes can then be merged back into the master branch. Git supports a few different workflow scenarios and we adopted to use the integration manager modal, where a single developer is responsible for committing all changes to the “blessed” repository.

Git is very good at doing reintegration merges, which is a huge benefit when you have developers working remotely.

For example, if a developer has been tasked with creating a new feature which is going to take several weeks to complete, and they don't have the facility to do reintegration merges as they work, then when they finally finish and try to push their new feature back to into the master branch it is likely that they will encounter lots of conflicts, as the master branch has probably moved forward with new features submitted by other developers, during those intervening weeks.

By using Git, the developer can fetch and merge the current state of the master branch into his feature branch daily, hence only needing to fix a few conflicts introduced in the previous 24 hours. When the developer has finished his work, and the feature branch is merged back into the master, Git can easily reintegrate the code common to the master branch, and just merge in the differences that make up the new feature with less chance of there being any major merge conflicts.

**Deciding when to start using Git**

As the integration manager or lead developer, you need to decide when in the system lifecycle you want to start using Git. It is recommended that you start using a version control system from the very start of a new project, even before you have released any software into a production environment.

When I am developing the initial version of a software product, I still use Git, I just commit everything to the master branch. Once the first version of the code is deployed to a production environment, then I start using the branching work flows.

You may decide only to start using version control after that first initial release of your software product, it's a judgement call you need to make based on your companies priorities.

**Deciding where the blessed repository should reside**

There are several different approaches you can adopt, and if you would like to have several copies of your blessed repository as a backup precaution, then there is nothing stopping you using several of the following options.

**Using a shared folder on an existing server**

The easiest and most convenient storage method is to use an existing server within your company. You would create a shared folder on this server, and use Window's Active Directory to set-up read and write permissions for each of the developers within your team.

As the integration manager, you would create the bare repository, and then move the repository folder structure to your shared folder on your server.

Once the initial project has been pushed to the blessed repository, the development team can then clone the project from this shared resource.

**Dropbox**

[Dropbox](https://www.dropbox.com/) is a cloud service that allows you to freely store and share up to 2Gb of data. When the client software is installed on multiple machines using the same account credentials, the software will sync the data in the Dropbox folder between all the machines.

You can create the blessed repository, and then move the repository folder structure to your Dropbox folder, so that it is replicated to all your developers machines.

You can [sign up](http://db.tt/GaQNRrnK) for a free Dropbox account and we will each get an extra 500 Mb of free storage.

**GitHub**

[GitHub](https://github.com/) is probably the best known online code hosting platform, and used by a large proportion of open source projects, thanks in part to offering unlimited public repositories.

GitHub has a very mature feature set offering team collaboration and code review tools, integrated issue tracker and Git powered wikis.

While the tools provided with GitHub are very good, if your working as part of a company and need private code repositories, then you're going to need to upgrade to one of paid plans.

**Bitbucket**

[Bitbucket](https://bitbucket.org/) is a secure online hosting solution for all your projects and repositories. Unlike GitHub it allows unlimited private and public projects for up to five team members. The paid plans only come into affect if you need to grant access to ten or more staff members. So if your software development team is small, then Bitbucket could be an attractive alternative.

All accounts come with unlimited disk storage and have a rather nice built in issue tracker, which allows you close issues and bugs automatically from your Git commit messages.

Bitbucket is run by the commercial company [Atlassian](http://www.atlassian.com/), which sells developer tools for agile project management and team collaboration.

**GitLab**

If you like the web interfaces offered by GitHub and Bitbucket, yet are uncomfortable having your companies source code held by another organisation or service, then maybe [GitLab](https://www.gitlab.com/) is the solution for you. GitLab is a self hosted solution which helps you to manage projects, issues, merge requests and easily browse your source code. It's a fast, secure and stable solution based on Ruby on Rails and Gitolite.

Being self hosted, you will need to install the software onto one of the supported Linux platforms, either [Ubuntu](http://www.ubuntu.com/) or [Debian](http://www.debian.org/). There is a complete step by step [installation guide](https://github.com/gitlabhq/gitlabhq/blob/stable/doc/installation.md) available, however if you are not happy with setting up your own Linux server, then there is a VMWare virtual machine appliance already compiled with the necessary components, available from [Turnkey Linux](http://www.turnkeylinux.org/gitlab)

Hosting company Digital Ocean has a simple one click install of GitLab, which means you can quickly spin up a virtual machine with a complete GitLab installation, ready to give it a try. If your busy and don't want to go through the installation process then consider signing up with [Digital Ocean](https://www.digitalocean.com/?refcode=2b0fd6f27444) (disclosure: that's a referral link; I'll get $10 towards my hosting if you sign up)

**Git GUI clients**

There are a couple of Git GUI clients worth considering, but if you can take the time to really learn the Git command line, you will have a transferable skill employers will want.

* [Smart Git](http://www.syntevo.com/smartgithg/) – is a really nice application to use if you're new to git. It is available in Windows, Mac OSX and Linux versions. One big benefit with this GUI application is how easily it integrates with remote GitHub repositories.
* [TortoiseGit](http://code.google.com/p/tortoisegit/) – is a simple implementation of Git management. While it does not have a GUI, it is a shell extension that merges as seamlessly as possible into the Windows Explorer context menu.

**Git Resources**

* Linus Torvalds, creator of Linux and Git, visits Google to talk to their staff about why Git was developed and why as a developer you should be using Git as your source code control system. The video is one hour ten minutes long, is very informative, and can be watched on [YouTube](http://www.youtube.com/watch?v=4XpnKHJAok8)
* Scott Chacon, who works at GitHub gives a very fast paced introduction to Git, available on [YouTube](http://www.youtube.com/watch?v=ZDR433b0HJY) the video is one hour twenty two minutes long. You can skip to time index 23.00 to start watching a live demo of branching and merging, and see the concept of reintegration merges being performed on a sample web site.
* [Mastering Git](http://tekpub.com/productions/git) by Rob Conery is a training video costing $15 USD. Over 3 hours of excellent content will give you the working knowledge that you will need to use Git confidently. Some of the key topics covered include branching, handling merge conflicts, rebasing and correcting mistakes. There is also useful segments on using Git with Visual Studio and Tips and Tricks. It's certainly worth looking at if you're the sort of person who absorbs more information by watching videos rather than reading tutorials.
* [Advanced Git](https://peepcode.com/products/advanced-git) is a 1 hour screencast by developer Ben Hoskings. It costs $12 USD and is money well spent as it will demystify the inner workings of Git. Ben covers some advance topics including the log, the commit graph, how to use the refheads and gives advice on merging and Git maintenance.
* [Git support within Visual Studio](http://www.hanselminutes.com/359/git-support-within-visual-studio), Scott Hanselman talks to Philip Kelley and Martin Woodward, developers on the project that's finally brought Git support to Visual Studio.

Other Git related links include:

* [A tour of Git: the basics](http://cworth.org/hgbook-git/tour/)
* [Manage source code using Git](http://www.ibm.com/developerworks/linux/library/l-git/)
* [A tutorial introduction to Git](http://www.kernel.org/pub/software/scm/git/docs/v1.2.6/tutorial.html)
* [Git FAQ](http://git.or.cz/gitwiki/GitFaq)
* [A Git core tutorial for developers](http://www.kernel.org/pub/software/scm/git/docs/v1.5.0.7/core-tutorial.html)
* [Git for the confused](http://www.gelato.unsw.edu.au/archives/git/0512/13748.html)
* [Git from the bottom up](http://www.newartisans.com/2008/04/git-from-the-bottom-up.html)
* [Using Dropbox as a Git repository](http://tumblr.intranation.com/post/766290743/using-dropbox-git-repository)
* [Using Git Diff to show the differences between two text files](https://nhs.io/diff/)

[↑ Top](https://nhs.io/git/#top)