Managing Development
Git and GitHub Basics

1. What is Git?

Git is software for managing different versions of code, and managing contributions from multiple developers.

Code is stored in a 'git repository'. When you make a change to some code, you write a short description of what the change is, and 'commit' this to the repository. Git records changes that have been made and the history of these commits.

2. What is GitHub?

GitHub is a website which hosts git repositories.

3. Datashield on GitHub

All datashield code is available at https://github.com/datashield

The code is split into a series of different repositories. There is a separate repository for each datashield R package.

4. Git Structure and Concepts

4.1 Repositories and branches

DataSHIELD exists as a series of R packages. Some installed on the computer of the researcher wanting to do analysis – these are the client packages; organised into the following repositories:
* opal
* dsBaseClient
* dsStatsClient
* dsGraphicsClient
* dsModellingClient

Some installed on each of the computers that holds the data to be analysed – these are the server packages; organised into the following repositories:
* dsBase
* dsStats
* dsGraphics
* dsModelling

One of git's selling points is that it makes non-linear development easier, so multiple people can work on different things and their edits can be merged together relatively painlessly. To accomplish this, git it built to make 'branching' very easy.

Each repository has a 'master' branch. This is the official, tested and production version of the code.

Each repository can have any number of other branches. These are copies of the master branch (or a copy of any other branch) from a given point. The purpose of the branches is to hold work in progress. Once a piece of work in progress has been finalised and tested, it can easily be merged into the master.

To be clear: each repository has its own branch structure. So each of the 9 repositories above has its own 'master' branch that is unique to it and independent of any other repository.

4.2 Local and remote

Each repository exists remotely and publicly on GitHub (Git calls github the repository 'origin').

When you get hold of a copy of the code to edit, you'll be getting the entire history of the repository including all of the branches that are on GitHub. You will have a local copy of everything.

Nothing you do locally will appear on GitHub unless you put it back there.

You can choose what goes to GitHub on a per branch basis – so there can be public branches, which exist on GitHub and in your local copy; and there can be private branches which only exist locally to you.

Your local copy of a branch (or multiple branches) might be 'ahead' of the copy on GitHub; because you've made some changes, but haven't put them back on GitHub. Or, your local copy of a branch or branches might be 'behind' the copy on GitHub; because someone else has made changes and put them on GitHub at sometime since you made your local copy. Or indeed, it could be both (which is when git can really help avoid problems).

5. Installing Git

5.1 Windows

There is a desktop application for Windows that integrates Git and GitHub together – 'GitHub for Windows'.

(1) Go to https://windows.github.com/ and download Github for Windows

(2) Run the installer
The installer might need to install some pre-requisites and reboot before finally being able to install github for windows.

(3) Sign in to the github client
Allow the github client to set your git configuration (name and email)

5.2 Linux

On Linux we only need to install Git itself. Use your package manager to install git. e.g.

```
$ sudo apt-get install git
```

Then tell git who you are:

```
$ git config --global user.name <insert name>
$ git config --global user.email <insert email
address>
```

6. Using Git

The two ways to use git covered here are:
(1) through the GitHub for Windows application,
(2) through the command line.

(1) is Windows specific, (2) applies to both Windows and Linux.

6.1 Cloning a repository

'Cloning' refers to the process of getting a copy of a repository. You only need to do this once.

GitHub for Windows
In the GitHub for Windows desktop application there is an option to clone repositories. As a member of the DataSHIELD organisation, you will be able to see a list of the DataSHIELD repos once you've signed in.

By default, cloning will create a folder with the same name as the repository you're cloning. So if you clone dsBase in the Documents folders, it will create a new folder Documents/dsBase which contains the code. By default GitHub for Windows clones into Documents/GitHub/<repo name>.

Command line / Git shell
Navigate to the folder you want to clone the repo into, for example:

```
$ cd Documents/GitHub
```

You need to know the web address of the repository you want to clone. However, they follow a consistent pattern of “<github url>/<username>/<repo name>.git”, for example:

```
$ git clone
https://github.com/datashield/dsBase.git
```

6.2 Pull a repository

Pulling refers to the process of updating your local copy of a repository. It is best to do this prior to making your own changes to make sure you are not a long way behind with any new changes.

GitHub for Windows
In the Windows desktop application, pulling is achieved by pressing the 'sync' button.

Command line / Git shell
Using the command line gives you a little more flexibility. Most simply, you can run:

```
$ git pull origin
```

Which will fetch and merge any updates to all of the branches that exist on GitHub (GitHub is the remote repository you are pulling from, which git calls the 'origin').

If you think that there are changes on your local version that are not on GitHub then you may want to use:

```
$ git pull --rebase origin
```

This will update your local version by putting your changes on top of any new changes from GitHub (as if the new changes from GitHub had happened first).

In practice it is better to first 'pull' to get up to date, and then make your changes. The --rebase option is useful when you forget to pull first.

You don't have to pull everything from GitHub, instead you can pull on a per branch basis. (And you can also use --rebase with this, too.) In which case use:

```
$ git pull origin <branchname>
```

Be careful – you will get into a mess if you pull a branch that is different from the branch you are currently working on.

6.3 Switching, creating and deleting branches

Remember, everything you do locally remains local unless you explicitly put it back on GitHub. You don't need to worry too much about accidentally deleting branches, or accidentally putting branches that you don't want to make public onto GitHub.

Switching branches
When you switch branches Git actually changes the contents of the repository folder. The contents of the folder always reflects whichever branch you are on.

So, if the 'add_more_files' branch contains 5 text files, but 'master' only contains 1 text file, then you can watch the four new files appear and disappear from the folder as you switch between branches. Similarly, you can watch the contents of a single file change as you switch between branches.

Git may ask you to commit changes you've made, before it will let you switch off a branch you've been working on. The state of the folder must, from Git's point of view, be 'clean'. It won't allow you to unexpectedly delete files by branch switching.

Creating branches
You can create a branch from any other (or from a tag or specific commit). Creating a branch basically gives you another copy of a set of commits. You may want to branch off master, off the most recent released version, or off some other branch.

Git makes branching easy. Branches give you a safe area in which to play with the code. You can do your work on branches, then merge back when you've perfected your changes.

GitHub for Windows

The branch you are working on, along with the other available branches, is displayed at the top of the application's window.

Clicking on the branch name gives you a menu for creating and performing other branch related tasks.

Command line / Git shell
To view all the available branches:

```
$ git branch -a
```

To switch from one branch to another:

```
$ git checkout <branch name>
```

To create a new branch, you specify the name of the new branch followed by whatever you want to make a copy of.

```
$ git checkout -b <new branch> <old branch>
```

To delete a branch:

```
$ git branch -d <branch name>
```

However, if Git detects that you are deleting a branch that hasn't been merged with another branch (i.e. it has changes that exist only on that branch), it will not let you delete it. Instead you must use:

```
$ git branch -D <branch name>
```

6.3 Add and commit your changes

Git allows you to keep track of the changes you've made to a repository. However, it only indexes things you tell it to keep track of. Moreover, you decide the size of each recorded change.

A commit is a record of changes since the last commit. When you commit, you are telling git to store the status of everything you've told it to keep track of.

It is good practice for commits to be small and logically independent. This is to give a clear history of what has changed. Also it makes it easier to see at what point problems may have been introduced, and to undo them without affecting other parts of the code.

GitHub for Windows
When you have made changes to files within a repository, the GitHub application will automatically display the fact you have 'uncommitted changes'.

If you click 'show', then you can see what has changed. By selecting some or all of the changed files, and by typing a message to describe the commit, you can commit your changes to the repository.

Remember, adding and committing is a local operation. Nothing goes to GitHub unless you 'sync' or 'publish' the branch.

Command line / Git shell
Once you've made a change, committing it is a two stage process.

Add the files you want to include in a particular commit:

```
$ git add <files you want to commit>
```

Or, if you want to include everything that's changed:

```
$ git add --all
```

Then, commit the changes you've just added:

```
$ git commit -m '<message describing the
changes>'
```

6.4 Merge your changes

Merging joins together two separate development histories. It allows you to bring changes from one branch into another.

With luck, merging should be straightforward. However, if exactly the same part of a file has changed on both branches, then there may be conflicts that need to be resolved manually.

GitHub for Windows
In the 'manage branches' dashboard, you can drag and drop two branches in order to merge one into the other.

Command line / Git shell
To merge a branch into another, first ensure you are currently working on the branch you want to insert the other branch into. For example, if you have been working on your own my-dev-changes branch, and want to merge your changes back into dev, then first switch back onto the dev branch:

```
$ git checkout dev
```

Now merge:

```
$ git merge my-dev-changes
```

This will bring the new commits on my-dev-changes into dev.

Git will help you with the resolution of any conflicts, but if you're unsure about what you're doing. You can always open a git shell and run:

```
$ git merge --abort
```

Which will take you back to how everything was before you ran the merge.

6.5 Push your changes

Once you're happy with the changes you've made and committed them, the next step is to push them to GitHub. (Pushing is the opposite of pulling).

If you didn't pull before making your changes and have therefore committed changes onto an out of date version of a branch, then git will not let you push those changes to

GitHub. Instead you must first pull, before you are allowed to push. In this situation the 'pull --rebase' can be very helpful.

If you have been working on a local branch that does not exist on GitHub then either you can merge this into a branch that does exist on GitHub and push that. Alternatively, you can push your local branch onto GitHub and make it public.

GitHub for Windows
Pushing is referred to as either 'syncing' or 'publishing' on the GitHub desktop application. If the application presents you with the option to sync, then the branch you've been working on already exists on GitHub. If the applications presents you with the option to publish, then that indicates you are working on a branch that only exists locally.

Command line / Git shell
To push a branch:

```
$ git push origin <branch name>
```

The command to delete a remote branch is actually a push command (as opposed to deleting a local branch, which is a branch command – see above).

To delete a remote branch, you add a colon in front of the branch name. (Yes, this is a bit odd.)

```
$ git push origin :<branch name>
```

7. Other Resources

GitHub Help Pages
https://help.github.com

Git Reference
http://gitref.org/

Everyday GIT With 20 Commands Or So
https://www.kernel.org/pub/software/scm/git/docs/everyday.html

Git Basics
https://git-scm.com/book/en/v1/Git-Basics

Git tips for beginners
http://www.markjberger.com/git-tips-for-beginners/

Super Quick Git Guide
https://wiki.archlinux.org/index.php/Super_Quick_Git_Guid

Git ready
http://gitready.com/

Undoing, fixing and removing commits in Git
https://sethrobertson.github.io/GitFixUm/fixup.html

Git tips and workflows
http://durdn.com/blog/2012/12/05/git-12-curated-git-tips-and-workflows/

A simple git branching model
https://gist.github.com/jbenet/ee6c9ac48068889b0912