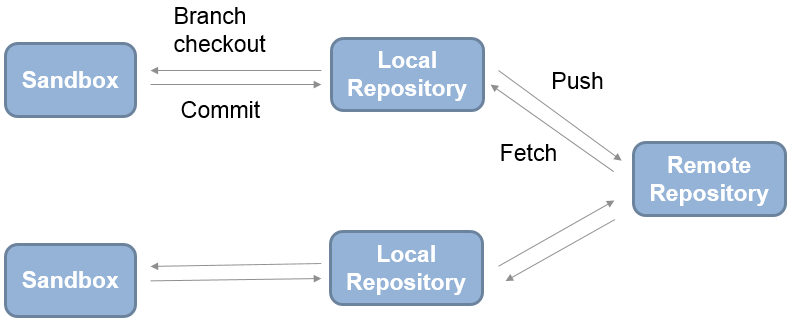
Source Control Using Git

# Using Git

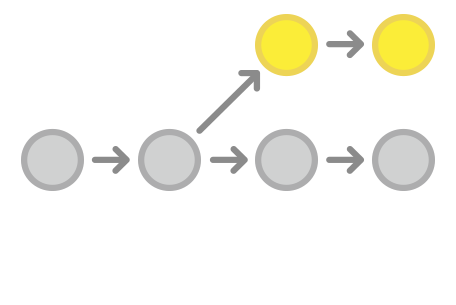
## High Level Explanation

* Source control is the management of changes to files. It is used to make collaboration on projects easier as multiple people can work on the same project at the same time then merge the changes together.
* Using source control one can merge and transfer changes from several users, roll back to previous versions, view changes made over time, and compare different changes and previous versions of the project.
* Definitions
  + Commit: Record of changes
  + Repository: A collection of files, folders and committed changes.
  + Remote Repository: An online repository (like a cloud service) which local repositories connect to.

## Workflow



* The remote repository is often hosted by a service such as Github or Gitblit. One is able to connect to a website where one can visualize the project’s changes, branches, and history. The remote repository could be kept on a local machine but one would miss out on these numerous advantages.
* The user can clone the remote repository and pull changes to a local repository. Pulling changes copies the files and merges them into what is currently on the local repository.
* Likewise, the user can push changes to the remote repository that have been made to the local repository. This will upload the changes and merge them into the remote repository.
* Within each repository are several branches. Each branch is created to work on a particular aspect of the project.



Branch 1

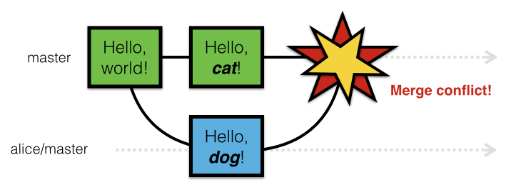
Branch 2

Create Branch

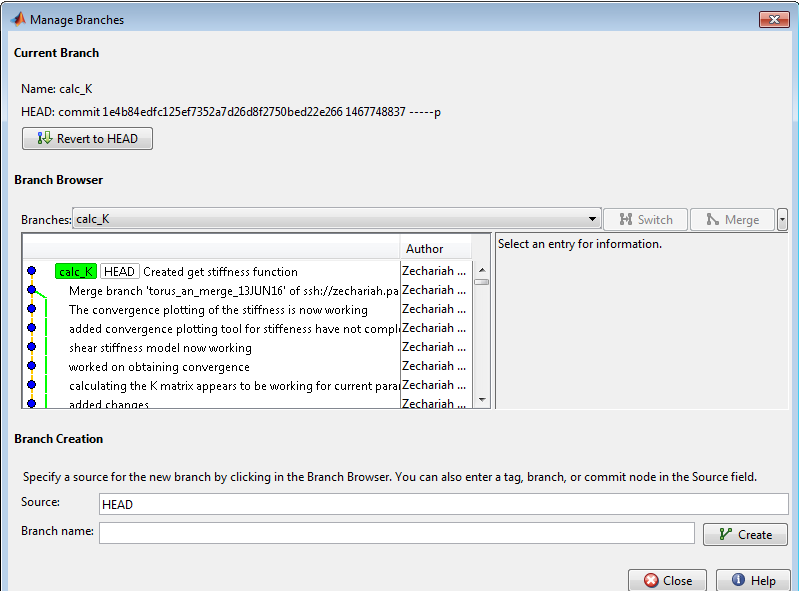
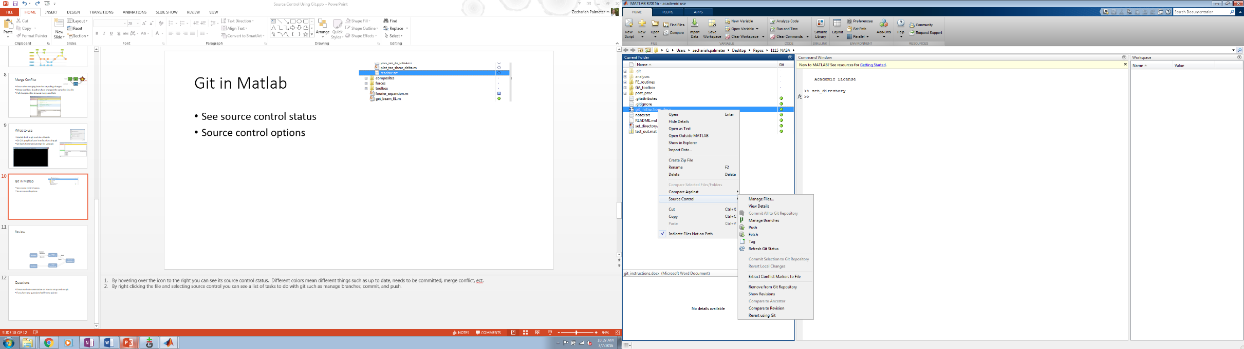
* Before working on a branch it must be checked out (switch between branches). Any changes you make to your project when a branch is checked out will be made to that branch. A branch can be checked out in git GUI. Checkout a different branch when you want to switch to working on a different feature. Below is a diagram of how the branches of a project might by organized.



Merge Branch

* When a feature is complete, you may want to merge branches to bring the changes from your completed feature into the main project. Before merging changes, all changes must be staged then committed. This means all changes you have made to your file must be added and recorded to your local repository. Try to make your commit messages as concise and descriptive as possible.
* When merging, something often happens known as a merge conflict. This is when a file has been changed in the same location by different users. To fix this, simply edit the file then add and commit the changes.

## Using Git in Matlab

* Matlab has built in integration with git source control. After a file has been added to a repository its current status can be seen in the Matlab current folder window. Green means it’s up to date with source control. Blue means there are uncommitted changes. Red means there is a merge conflict.
* By right clicking on the file and selecting source control, the file can be reverted to a previous version, committed to the local repository, and compared to a previous version. One can also manage branches and push changes.

## Using Git GUI

* In the repositories tab, one can see the history of the repository.
* In the bottom right hand corner, changes can be staged and committed with a commit message.
* Under the branches tab, branches can be checked out, created and deleted.
* Under the merge tab branches can be merged. To pull changes from the remote, select merge with tracking branch. Abort merge removes any uncommitted changes from the repository.

## Git Best Practices

* Pull changes frequently to prevent merge conflicts (how frequently depends on the amount of people working on the project, with more people one will want to pull more frequently).
* Create a new branch for each feature you want to add to the project.
* Have a good procedure for approving branches to be merged. (Some hosting websites such as Github make this process easy with tools such as pull requests.)
* Always have descriptive and concise commits that answer the questions: What was the motivation for the change? How does it differ from the previous implementation?
* Commit related changes and commit often, but don’t commit half done work

# Git Setup

## Download

1. Download git at <https://git-for-windows.github.io/>
2. In the section on adjusting your PATH, choose the install option to “*Use Git from the Windows Command Prompt*.” This option adds Git to your PATH variable, so that MATLAB can communicate with Git.
3. In the section on configuring the line-ending conversions, choose the option “*Checkout as-is, commit as-is”* to avoid converting any line endings in files.
4. Use the default for all other settings when installing git.

## Set Up ssh key Pair

* Obtain access to a remote repository hosted on Gitblit, Github, ect.
* Open git bash and type “ssh-keygen” then follow the prompts to create a ssh key.
* Navigate to the newly created .ssh folder and open id\_rsa.pub with notepad and copy all its contents.
* Go to your online remote repository (this will be a websight) and in profile settings, there will be an option to add an ssh key. Select this and paste your copied ssh key into the prompted text field then save it.

## Set Up Local Repository

1. Navigate to your online remote repository websight and find the url of the repository. Copy this url. If possible, copy the ssh:// url as this will allow easier operations in the future.
2. Create a new folder where you would like your various project files to be kept.
3. Right click and select open git gui. (Note: this documentation will continue using only the git gui. All proceeding operations can be done using a terminal such as git bash. At the end of the tutorial are commands for using the terminal.)
4. Select clone repository and paste the copied url in the source location field. In the target directory paste the path to your newly created folder. At the end of this file path add a forward slash then the name of a new folder where the specific project files from the remote repository will be kept. Click clone.
5. Your local repository now is linked to the remote repository and matches the current state of the remote repository.

## Add .gitattributes and .gitignore files

1. After installed, If you do not already have a .gitattributes file in your sandbox folder, create one at the MATLAB command prompt by typing: edit .gitattributes
2. Add these lines to the .gitattributes file:

\*.mat -crlf -diff -merge

\*.p -crlf -diff -merge

\*.slx -crlf -diff -merge

\*.mdl -crlf -diff –merge

\*.mdlp -crlf -diff -merge

\*.slxp -crlf -diff -merge

\*.sldd -crlf -diff -merge

\*.mexa64 -crlf -diff -merge

\*.mexw64 -crlf -diff -merge

\*.mexmaci64 -crlf -diff -merge

\*.xlsx -crlf -diff -merge

\*.docx -crlf -diff -merge

\*.pdf -crlf -diff -merge

\*.jpg -crlf -diff -merge

\*.png -crlf -diff –merge

1. Save the file in the repository folder
2. Repeat the same process and create a .gitignore file with the file extensions you want source control to ignore using the same notation \*.file\_extesnsion
3. Restart Matlab

# Git Commands for Git Bash

|  |  |
| --- | --- |
| **CREATE** |  |
| Clone an existing repository | $ git clone ssh://user@domain.com/repo.git |
| Create a new local repository | $ git init |
|  |  |
| **LOCAL CHANGES** |  |
| Changed files in your working directory | $ git status |
| Changes to tracked files | $ git diff |
| Add all current changes to the next commit | $ git add . |
| Add some changes in <file> to the next commit | $ git add -p <file> |
| Commit all local changes in tracked files | $ git commit -a |
| Commit previously staged changes | $ git commit |
| Change the last commit (Don't amend published commits!) | $ git commit --amend |
|  |  |
| **COMMIT HISTORY** |  |
| Show all commits, starting with newest | $ git log |
| Show changes over time for a specific file | $ git log -p <file> |
| Who changed what and when in <file> | $ git blame <file> |
|  |  |
| **BRANCHES & TAGS** |  |
| List all existing branches | $ git branch -av |
| Switch HEAD branch | $ git checkout <branch> |
| Create a new branch based on your current HEAD | $ git branch <new-branch> |
| Create a new tracking branch based on a remote branch | $ git checkout --track <remote/branch> |
| Delete a local branch | $ git branch -d <branch> |
| Mark the current commit with a tag | $ git tag <tag-name> |
|  |  |
| **UPDATE & PUBLISH** |  |
| List all currently configured remotes | $ git remote -v |
| Show information about a remote | $ git remote show <remote> |
| Add new remote repository, named <remote> | $ git remote add <shortname> <url> |
| Download all changes from <remote>, but don‘t integrate into HEAD | $ git fetch <remote> |
| Download changes and directly merge/integrate into HEAD | $ git pull <remote> <branch> |
| Publish local changes on a remote | $ git push <remote> <branch> |
| Delete a branch on the remote | $ git branch -dr <remote/branch> |
| Publish your tag s | $ git push --tags |
|  |  |
| **MERGE & REBASE** |  |
| Merge <branch> into your current HEAD | $ git merge <branch> |
| Rebase your current HEAD onto <branch> Don‘t rebase published commits! | $ git rebase <branch> |
| Abort a rebase | $ git rebase --abort |
| Continue a rebase after resolving conflicts | $ git rebase --continue |
| Use your configured merge tool to solve conflicts | $ git mergetool |
| Use your editor to manually solve conflicts and (after resolving) mark file as resolved | $ git add <resolved-file> |
|  |  |
| **UNDO** |  |
| Discard all local changes in your working directory | $ git reset --hard HEAD |
| Discard local changes in a specific file | $ git checkout HEAD <file> |
| Revert a commit (by producing a new commit with contrary changes) | $ git revert <commit> |
| Reset your HEAD pointer to a previous commit and discard all changes since then | $ git reset --hard <commit> |
| …and preserve all changes as unstaged changes | $ git reset <commit> |
| …and preserve uncommitted local changes | $ git reset --keep <commit> |
|  |  |
| **Help and Documentation** |  |
| Get help on the command line | $ git help <command> |

# Additional Resources

Good explanations of most git features: <https://www.atlassian.com/git/tutorials/>

Matlab Documentation: <http://www.mathworks.com/help/matlab/matlab_prog/set-up-git-source-control.html> and <http://blogs.mathworks.com/community/2014/10/20/matlab-and-git/>

Git GUI: <https://nathanj.github.io/gitguide/tour.html> and <https://www.youtube.com/watch?v=-t5vaN7fEIE>

Video Explanation of git using git bash: <https://youtu.be/Y9XZQO1n_7c>