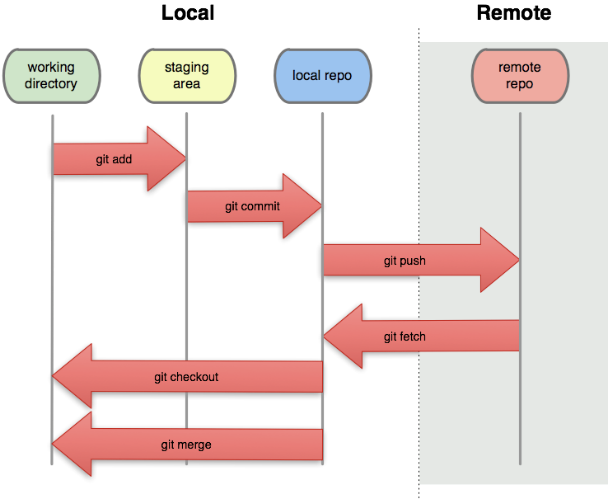
Version Control System (VCS) – Git

The idea of a version control system is to control and keep track of the different versions of code written in a project, providing the ability for many people to work on a project at the same time, to rollback software changes, and keep intellectual property which may no longer be required in the current version of a project.



Initialising a Repository

For each project which is being worked on, use “git init” to initialise a repository in the working directory and create a hidden .git directory to track changes. Undo this using “rm -rf .git”

Add file to staging area

To tell git what files are to be worked on and tracked in the repository, add them to the staging area using “git add <filename>”. Staging allows for files to be committed in specific batches, instead of all at once. For example if three files are changed during work on a project then the two related files could be staged and committed at a different time to an unrelated third file. Using “git add .” will add all files in working directory to staging.

It is possible to revert changes to files which have been modified by checking them out of the git repository again. This will return the file to the one last committed.

If a file is staged, it can be unstaged using “git reset HEAD <filename>”, however note that this does not reset the already modified file, just removes it from staging.

When moving files using the git command “git mv <filename> <destination>”, they are automatically staged. Otherwise when each file is moved it will have to be staged separately.

Committing files

Committing a file is similar to saving a checkpoint of it. Commit files by “git commit -m “<comment>””, where the comment is the commit message (name) defined by -m. If no commit message is added then git will open a text editor to allow for a message to be added.

Commits can be amended by staging the updated files and using “git commit --amend -m “<comment>”. This will update the most recent commit to include the changes.

Undoing commits

It is possible to revert a commit using “git revert <commit-hash-value>”. This will allow a comment to be added and revert the commit to the previous state. If you revert to a commit earlier in the tree, you may have to resolve some conflicts and recommit the files.

Resetting a commit

It is possible to reset a repository to a specific commit using the reset command. When given a commit reference (i.e. a hash, branch or tag name), the reset command will

* Rewrite the current branch to point to the specified commit
* Optionally reset the staging area to match the specified commit
* Optionally reset the working directory to match the specified commit

Using “git reset --hard <commit-hash>” will reset the repository and move the master to the hash specified, removing the history from the immediate log. The full history can still be seen in “git log --all”.

Check status

Using “git status” shows: the branch, if there are any new commits to be performed, and whether there are any files in the working tree not yet committed. It is good practice to check the git status before staging or committing any files, as to check that the repository is in the state you expect it to be in.

Viewing change log

Using “git log” will list all information about the previous commits in the repository. Using “git log --pretty=oneline” will make each commit one line with only the hash and comment. There are also many other ways to format the log such as “—since=’5 minutes ago’”. The best format so far is

* git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

Removing files from tracking

Simply deleting files from a repository does not stage and commit (track) a delete. To delete a file which has previously been committed use and delete the file use “git rm -f <filename>”, else use “git rm --cached <filename>” to remove the file from tracking without deleting it. It is then possible to add the file to a .gitignore file to stop any future tracking.

Returning to a previous branch state

Go into the git log and find the commit has of the state which is to be recovered. Then perform “git checkout <hash#>” to recover to the state 2 and create a “detached HEAD” state.

Returning to the master branch state

To return to the most recent main master branch state use “git checkout master”.

Tagging Commits

Use “git tag <tagname>” to tag the current commit the HEAD is at making it easier to refer to than the hash value for it. All tags which have been made can be seen using “git tag”. Git tags will also been seen in the git log. Tags can be deleted using “git tag -d <tagname>”

Dumping on git

Use “git cat-file -p <hash/treehash/libhash/filehash>”

CreatingLink a new branch

In an existing local repository use “git checkout -b <branchname>” to create a new branch in the same repository. If the branch already exists git use “git checkout <branchname>” to switch. To then push the new branch to to one on github use “git push -u origin greet”, which will then create a new branch on github.

When branches diverge, using --graph and --all on the git history log will show each branch diverging. To see all branches being tracked use “git branch -a”.

Merging branches

Using “git merge <branchname>” will merge the branch the HEAD is currently at with the branch defined in branchname. If there are no conflicting files, the difference will simply be added, however if there are conflicts, use “git status” to see the conflicted files, then edit and resolve each conflict before staging and committing the files again to complete the merge.

Tracking branches

Rebasing

Similar to merge but applies the commits of the rebased branch into the history of the modified branch. “git rebase <branchname>”. Since rebase rewrites the commit history, do not use when the exact history is required, or when the repository is shared with others. Rebasing when the repository is linked to github will create conflict.

Cloning repositories

Repositories can be cloned locally or via github using “git clone <localname> <clonedname>” and “git clone Use “git remote add origin git@github.com:cpcwood/<repositoryname>.git” respectively.

Using “git remote show origin” will show the origin for the cloned repository, either being local or online.cd

A repository can be cloned without the working directory by “git clone --bare <reponame> <newreponame>.git” this will simply clone the .git folder of the repository giving access to the history and master commits. A bare repository is usually used for sharing a repository.

However, when directly cloning someone elses repository directly from github, in which you are not a contriubuter (no push access), you will lose the abiltity to push back to github. Therefore, to get around this first ‘fork’ the repository to your own github, then clone the forked reposistory to your local machine. This will set up a remote to the forked reposistory, then a pull request can then be made on github for any commits which you want to merge with the original repository.

Updating cloned repository

Using “git fetch” will pull new commits from the origin of the cloned repository, however it will not apply them to the branches straight away. To apply changes you must merge them using “git merge origin/master”. If you want to combine these actions, simply use “git pull”. When cloning a repository, only the master branch will clone, other branches can be seen using “git branch -a”. to add a to tell the local branch which will track a repository that it is linked to one on github, then check the set up with “git remote branch which wasn’t cloned, use “git branch --track <branchname> origin/<branchname>” -v”

Pushing changes to a shared repository

Create a remote inPush local code to github

Then push the code from the local repository to the shared .git repository, called shared “git remote add shared <repositoryname>.git”. Then push files online one using “git push shared master” (similar to pushing to github using the “git push-u origin master”).”. This can be also donepushes the opposite way round for pulling branches of other repositories. To do this add a remote into the repository linking it to the repository to pull from called shared, then track the origin master branch to pullof the repository online, and pull the -u saves the settings so only “git push” is required to push the master branch next time.

Setting up a git server

Use “git daemon --verbose --export-all --base-path=.” from the working directory to set up a daemon in the localhost, then copy the repository across using “git clone git://localhost/hello.git network\_hello”. To push to a network daemon add --enable=receive-pack to the daemon command, however there is no authentication on the daemon so anyone can push to it.

Renaming git repository

Rename repository on github, then change folder name on computer and use the following command to update the remote connection to github: ‘git remote set-url <new-git-url>’

Alias’

Ways to shorten typing on common commands. Add to global .git config file

[alias]

co = checkout

ci = commit

st = status

br = branch

hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

type = cat-file -t

dump = cat-file -p

Pull code from github

To pull the current code from github use “git pull origin master”

Cloning repository from github

Navigate to the repository that is to be cloned on github, then click clone by ssh. This will provide the repository location. Then in cmd navigate to desired projects folder and use “git clone <repositorylocation>” and the repository will be cloned into a new folder. The clone function automatically initiates the repository, links it, and pulls the data.

Ignore files on git

To make the git version control software ignore files, create a .gitignore and add the file names which are to be ignored into it. To make the .gitignore file ignored, it must also be added to the ignore list.

README.md

The clue is in the name. A README is a file that should be read carefully before using any piece of code (as in, READ! ME!). At the barest minimum, it should contain a description of what the code does, how to install it, how to use it and how to run its tests. In addition, READMEs often include other information such as:

* Configuration instructions
* A file manifest (list of files included)
* Copyright and licensing information
* Contact information for the distributor or programmer
* Known bugs
* Troubleshooting
* Credits and acknowledgments
* A changelog (usually for programmers)
* A news section (usually for users)

So, as a minimum, here's what you need:

* A README.md file in the root folder of your repo
* The title of the project
* What the project is/what it does
* How to install it (what to clone, what to run to get all dependencies)
* How to run it (is it a command line tool? Do you have to load it into IRB? Is is a web application? What port needs to be used?)
* How to run the tests