git for team projects: masterclass

Team Project 2022-23

In this session, we will try out some of the basic features of git. You are expected to follow along an try the commands with your own test repository.

This session will use the terminal, however the principles are the same as if using an IDE or GUI.

This session is not exhaustive. Many of the commands are rarely used and it is completely fine to consult the documentation or a git cheat-sheet when needed.

1 git basics: creating repositories, SSH keys, clone, add, push

1.1 creating a repository online

You will be given a team repository on `https://git.cs.bham.ac.uk` That repositiory will be sent to you by IT. We have created a sample reporsitory: `https://git.cs.bham.ac.uk/shahs/sample-git-project`.

You may also use an online service to create a repository.

1. go to `https://git.cs.bham.ac.uk` and login
2. click on the "new project" button (top right)
3. choose "create a blank project"
4. give the project a name, like "example-project"
5. (optionally - add users, e.g. a teammate to your project)

1.2 generate and add an ssh key to your CS repository

For this session we will create an example repository on `https://git.cs.bham.ac.uk`. In order to pull or push to the repository on `https://git.cs.bham.ac.uk` (or `https://git-teaching.cs.bham.ac.uk`) we will first need to generate and add an SSH key to the server.

Using an SSH key with git considered best practice. It is more secure and more convenient than entering your password every time, although it requires the some one-off setup: creating an key and adding it to the server.

Full instructions on how to create SSH keys on your computer can be found here: `https://docs.gitlab.com/ee/ssh/` or here : `https://linuxkamarada.com/en/2019/07/14/using-git-with-ssh-keys/`. The below instructions should work on a mac or linux.

The command to generate an rsa 2048 bit key is:

```
ssh-keygen -t rsa -b 2048 -C "example key for masterclass"
```

You can add a passphrase to unlock the key or not, depending on how secure your machine is and how comfortable you feel.

The command to get the public key is:

```
cat ~/.ssh/id_rsa.pub
```

where `id_rsa.pub` is the file where the public key was created.

Now copy that public key to the page:

```
https://git.cs.bham.ac.uk/-/profile/keys
```

on your machine do the following:

```
ssh-add ~/.ssh/id_rsa
```

to add the generated key to your local machine's ssh client.

1.3 cloning an online repository

- `git clone <repo-url> <directory>`
  or `git clone <repo-url>`
  Creates a clone of repository under the given directory. If the directory is omitted, Git creates the clone in a folder with the same name as the repository.
- Get the ssh clone url from the repository clone button. If you are using an ssh clone url and have set up keys, you will not be asked for your username or password.

1.4 your first commit

- (create a file)
- `git add <filenames>`
  Adds the given files to the index
- `git commit -m "Message"`
  Creates a new snapshot in the object store based on the files that have been added to the index The message should be meaningful and should describe what changed as it will be associated with the snapshot If -m "Message" is omitted, Git will start up an editor for you to type your message into manually.
- `git pull`
  before pushing, get into the habit of doing a pull first, there maybe remote changes made by other members of your team that need to be merged manually.
- `git push`
  Pushes snapshots from your local object store to an object store of a remote repository. If the repository was created using git clone then git push will push back to that

1.5 write useful git commit messages, make small changes and use a .gitignore

The clearer and shorter your commit messages are, the easier it will be for the rest of your team to understand what has changed in the repository.

If you find you are writing very long commit messages with several changes, stop and instead make the changes over several commits. This makes understanding the changes much easier and more testable, and also easier to pinpoint and revert when something goes wrong.

In many programming languages, intermediate files are generated during testing and build that should not be committed to a repository. These could be executables, log files or any kind of temporary files. Creating a `.gitignore` file will allow you to specify which files or directories to ignore. There are even repositories of .gitignore files online for most languages and frameworks.

1.6 make a non-git, local folder into an online git repository

In some situations, you may want to create an online repository from some code or a project you have worked on locally. To upload some local code, you must `git init` and git remote add, as well as a few extra tasks to be aware of.

- `git init --initial-branch=main`
  Creates a git repository structure from the current directory, where the current directory is not already a repository. main is the name of the branch.
- `git remote add origin <repo-url>`
  Adds a new remote location for the repository. `origin` here is the conventional name for the remote repository.
- `git add -A`
  Add the files in the current folder (the -A means add all files recursively)
- `git pull`
  `git commit -m "initial commit"`
  `git push -u origin main` Adds a new remote location for the repository. note that here we make it clear that we are pushing to the `main` branch of the `origin` repository.

n.b. once you have pushed the code it is strongly recommended to `git clone` the repository into another directory so you get the meta-data from the server.

1.7 useful general commands

- `git status`
  Shows the status of the project: which files in the working directory have changes not recorded in the index, which branch you are working on, and which files are not tracked by Git
- `git diff`
  Shows what has been created or changed between the last commit and any local, uncommitted edits.
- `git log`
  Shows the snapshots in the object store as git commit log messages
- `git checkout <commitID>`
  Will undo changes you have made to files in your working directory, reverting the files back to the version in commitID. commitID is a reference t a commit. commitID which can be found using `git log`

2 more advanced git: resolving conflicts, branching

2.1 resolving merge conflicts

Git merge : resolving conflicts example

```
Auto-merging some file.py CONFLICT (content): Merge conflict in some file.py Automatic merge failed;
fix conflicts and then commit the result.
```

the conflicting files will have a section:

```
<<<<<<< HEAD:some file.py
here you have the version on main branch because main is the branch where you ran the merge command
=======
here you have the version from the feature branch, the one you are trying to merge
>>>>>>> feature:some file.py |
```

- `git status`, will show:
  `both modified: some file.py`
- Git adds standard conflict-resolution markers (diff) to the files that have conflicts
- To fix this, keep only the code for the master branch, remove the markers, and then run `git add` on the file(s)

2.2 creating branches

- `git branch example-feature`
  Creates a new branch (called example-feature here) but does not switch to that branch
- `git checkout -b example-feature`
  Creates a new branch (called example-feature here) and switches to it at the same time
- `git checkout example-feature`

  - Switches to the example-feature branch
  - If you are on a different branch and want to switch back to main
  - Run `git checkout main`
  - main is a conventional name for the main branch

2.3 merging branches

First, make sure you do not have any uncommited local changes before merging - do a `git commit`.

- `git checkout main`
  Switches to the main branch Git resets the working directory to look like it did the latest commit on main
- `git merge example-feature`
  Merges the example-feature branch into the current branch (main in case we switched to the main branch earlier)
- `git pull`, `git commit -m "merge message"` , `git push` the merged branch

2.4 merging branches - branch merge conflicts

Merging branch conflicts is similar to merging normal conflicts.

As an exercise, if you were able to complete all of the above try to create a branch merge conflict with one or more teammates.

3 git best practice

- Use a descriptive commit message
- Make each commit a logical/atomic unit (as much as you can)
- Test your changes before committing them
- Share (push) and get (pull) changes frequently
- pull before push
- Don't commit generated files (`.gitignore`)
- Keep short lived branches
- Never store passwords or private key files in a repository - git history means these can never be deleted!
- Don't commit unfinished work (week three: CI/CD)