GitHub RDP Session

Sajid Fadlelseed
P.h.D. Student

School of Physics, Engineering and Computer Science (SPECS)

June 15, 2022

Overview

Overview

- keeps track of your creative output
- tracks what is changed
- tracks who made the changes
- tracks why changes were made
- everyone needs it
  - developers / designers
  - writers / producers
  - artists / composers
- you can use it as part of team
- you can use it by yourself

Why version control

- history: change-by-change log of your work
- fearless experimintation: easy to create a "sandbox" to try new things
- synchronisation - easy to keep team members always up-to-date
- accountability - know who made each change and why
- conflict detection - keep the build clean every time

You already do version control
even if you don't use software

- `save as` . . . for backup individual files
- duplicate or compress directories
- sometimes you save them it on Dropbox, GoogleDrive etc . . .

Advantages

- backups - every version is kept around
- change tracking - commit messages let you know why things changed
- rollback to previous versions - similar to undo
- labelling significant changes - tags/labels identify the state of the source that matches each release

History

- stand-alone and file-focused
  - 1972 Sourec Code Control System (SCCS)
    - ★ Unix only
  - 1982 Revision Control System (RCS)
    - ★ cross-platform,
- centerlised
  - 1986 Centerlised Version Control (CVS)
    - ★ first central repository
    - ★ file focused
  - 2000 SubVersioN (SVN)
    - ★ non text files
    - ★ track directory structure
- distributed
  - 2005 Git:
    - ★ created by Linus Torvalds after BitKeeper become commercial.
    - ★ broadly used in conjuncation with GitHub which offers free hosting for open-source projects

Centralised vs Distributed

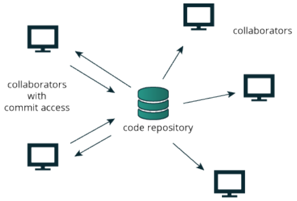


Figure: Centralised Version Control System

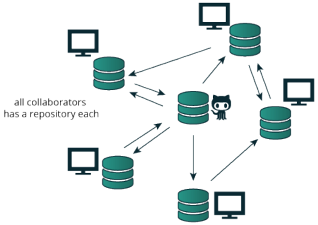


Figure: Decentralised Version Control System

Installing Git

1. go to https://git-scm.com/downloads
2. download the software for Mac/Linux/Windows
3. install Git choosing all of the default options

Required Commands We'll be using the following terminal commands:

- `ls` - list files and directories
- `mkdir` - create a new directory
- `cd` - change directories
- `rm` - remove files and directories
- `pwd` - print working directory

First time Git configuration

- setup your name
  ```
  git config --global user.name "your-name"
  ```
- setup your email
  ```
  git config --global user.email "your-email"
  ```
- make sure that Git output is colored
  ```
  git config --global color.ui auto
  ```

Link Git with your text editor

- Sublime Text Setup
  Unix/Linux
  ```
  git config --global core.editor "'/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl' -n -w"
  ```
  Windows
  ```
  git config --global core.editor "'C:/Program Files/Sublime Text/sublime\_text.exe' -n -w"
  ```
- Atom Editor Setup
  Windows/Unix/Linux
  ```
  git config --global core.editor "atom --wait"
  ```
- VSCode Setup
  Windows/Unix/Linux
  ```
  git config --global core.editor "code --wait"
  ```

Create directories

- create a directory called `rdp`
- use the cd command to move into the `rdp` directory
- inside `rdp` create another directory called `new-project`
- use the cd command to move into the `new-project` directory

```
mkdir rdp
cd rdp
mkdir new-project
cd new-project
```

Create a Git repository

- Create brand new repository from scratch on your computer using command

```
git init
```

- Copy an existing repository from somewhere else to your computer using command

```
git clone
```

- Check the current state of the repository

```
git status
```

Create a Git repository cont.

- Running the `git init` command sets up all of the necessary files and directories.

```
git init
```

- All of these files are stored in a directory called `.git` this directory is the "repo"! here is where git records the file changes and keeps track of everything!

WARNING

Don't edit any file inside the `.git` directory. This is the heart of the repository. If you change file names and/or file content, Git will probably lose track of the files that you're keeping in the repo, and you could lose a lot of work! It's okay to look at those files though, but don't edit or delete them.

Create a Git repository cont.

`.git` contents are:

description file	this file is only used by the GitWeb program, so we can ignore it
hooks directory	this is where we could place client-side or server-side scripts that we can use to hook into Git's different lifecycle events (Remember, other than the "hooks" directory, you shouldn't mess with any of the content in here. The "hooks" directory can be used to hook into different parts or events of Git's workflow, but that's a more advanced topic that we won't touch it in this workshop)
info directory	contains the global excludes file
objects directory	this directory will store all of the commits we make
refs directory	this directory holds pointers to commits (basically the "branches" and "tags")

Useful Links:

https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain

https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

Clone an existed repo

- Copy an existing repository from somewhere else to your computer using command

  `git clone <path-to-repository-to-clone>`

```
git clone https://github.com/nvas/cloned-project.git
```

- Running the git clone command will:
  1. take the path to an existing repository;
  2. by default will create a directory with the same name as the repository that's being cloned;
  3. can be given a second argument that will be used as the name of the directory;
  4. will create the new repository inside of the current working directory.

Useful Links:
Cloning an Existing Repository:
https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository#Cloning-an-Existing-Repositoryhttps://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
git clone docs: https://git-scm.com/docs/git-clone

git clone Tutorial: https://www.atlassian.com/git/tutorials/setting-up-a-repository

Status update

- To figure out what's going on with a repository use `git status`

```
git status
```

- At this point, we have two Git repositories:
  1. the empty one that we created with the `git init` command;
  2. the one we cloned with the `git clone` command.
- Compare `git status` output between the two repos.

Knowing the status of a Git repository is extremely important.

Review existed repository

- Display information about the existing commits using `git log`

```
git log
```

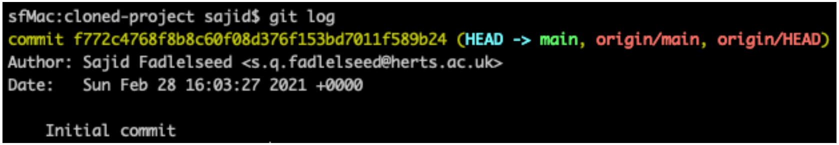
  - to scroll down
    1. press `j` or `↓` to move down one line at a time
    2. press `d` to move by half the page screen
    3. press `f` or `space-bar` or the `Page Down` button to move by a whole page screen
  - to scroll up
    1. press `k` or `↑` to move up one line at a time
    2. press `u` to move by half the page screen
    3. press `b` or the `Page Up` button to move by a whole page screen
  - press `q` to quit out of the log (returns to the regular command prompt)

- Display information about a commit using `git show`

```
git show
```

Review existed repository cont.

- Running the `git log` command displays all of the commits of a repository. By default, this command displays:
  - SHA
  - author
  - date
  - message

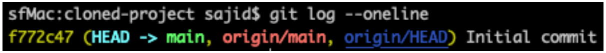


```
sfMac:cloned-project sajid$ git log
commit f772c4768f8b8c60f08d376f153bd7011f589b24 (HEAD -> main, origin/main, origin/HEAD)
Author: Sajid Fadlelseed <s.q.fadlelseed@herts.ac.uk>
Date:   Sun Feb 28 16:03:27 2021 +0000

    Initial commit
```

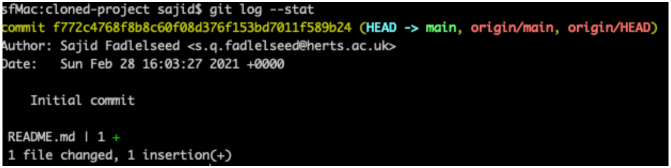
Review existed repository cont.

- lists one commit per line using `git log --oneline`
- shows the first 7 characters of the commit's SHA
- shows the commit's message



```
sfMac:cloned-project sajid$ git log --oneline
f772c47 (HEAD -> main, origin/main, origin/HEAD) Initial commit
```

- displays the file(s) that have been modified `git log --stat`
- displays the number of lines that have been added/removed
- displays a summary line with the total number of modified files and lines that have been added/removed



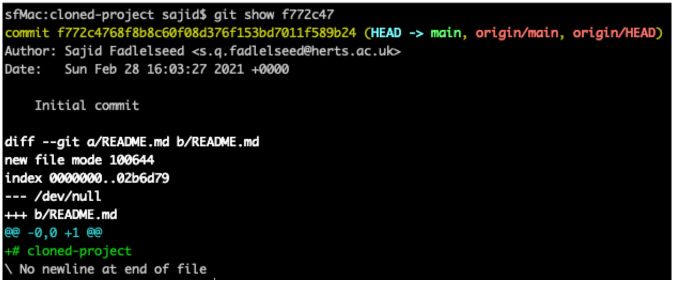
```
sfMac:cloned-project sajid$ git log --stat
commit f772c4768f8b8c60f08d376f153bd7011f589b24 (HEAD -> main, origin/main, origin/HEAD)
Author: Sajid Fadlelseed <s.q.fadlelseed@herts.ac.uk>
Date:   Sun Feb 28 16:03:27 2021 +0000

    Initial commit

 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Review existed repository cont.

Show a specific commit info using command `git show <SHA>`



```
sfMac:cloned-project sajid$ git show f772c47
commit f772c4768f8b8c60f08d376f153bd7011f589b24 (HEAD -> main, origin/main, origin/HEAD)
Author: Sajid Fadlelseed <s.q.fadlelseed@herts.ac.uk>
Date:   Sun Feb 28 16:03:27 2021 +0000

    Initial commit

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..02b6d79
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# cloned-project
\ No newline at end of file
```

- the commit
- the author
- the date
- the commit message
- the patch information

Add commits to a repository

- Add files from the working directories to the staging index

`git add`

- Take files from the staging index and save them in the repository

`git commit`

- Display the difference between two versions of a file

`git diff`

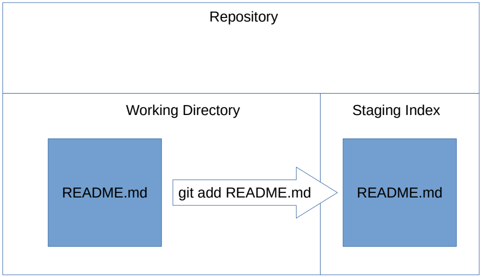
Add commits to a repository cont.

- First, create a file named README.md, and fill it with some data:

```
touch README.md; echo "data" > README.md
```

- the git add command is used to move files from the Working Directory to the Staging Index

```
git add README.md
```



Make a commit

- Link Git with your text editor (see slide No. 8)
- The `git commit` command takes files from the Staging Index and saves them in the repository.

```
git commit -m "First-commit"
```

- Then, use `git log` to review the commit you just made!

Tagging, branching and merging

- Add tags to specific commits e.g. this is the beta release

  `git tag`

- Allows multiple lines of development

  `git branch`

- Switch between different branches and tags

  `git checkout`

- Combine changes on different branches

  `git merge`

The tag command

Used to add a marker on a specific commit and the tag does not move around as new commits are added. This command will:

- add a tag to the most recent commit

```
git tag -a beta
```

- add a tag to a specific commit if a SHA is passed

```
git tag -a beta a87984
```

- A tag can be deleted with the `-d` flag (for delete!) and the name of the tag:

```
git tag -d beta
```

- NOTE: In the command above `git tag -a beta` the `-a` flag is used. This flag tells Git to create an annotated flag. If you don't provide the flag (i.e. git tag v1.0) then it'll create what's called a lightweight tag. Annotated tags are recommended because they include a lot of extra information such as:
  - the person who made the tag
  - the date the tag was made
  - a message for the tag

The branch command

Used to interact with Git branches

- Lets you make changes that do not affect the master branch
- It can be used to
  - list all branch names in the repository

    ```
    git branch
    ```
  - create new branches

    ```
    git branch <new_branch_name>
    ```
  - delete branches

    ```
    git branch  -d <branch_name>
    ```
  - delete remote branches

    ```
    git push <remote_name> --delete <branch_name>
    ```

The checkout command

```
git checkout <branch_name>
```

It's important to understand how this command works. Running this command will:

- remove all files and directories from the Working Directory that Git is tracking (files that Git tracks are stored in the repository, so nothing is lost)
- go into the repository and pull out all of the files and directories of the commit that the branch points to

Switch and create branch in one command

```
git checkout -b <new_branch_name>
```

The merge command

- Combining branches together is called merging.

```
git merge <name-of-branch-to-merge-in>
```

- Fast-forward merge – the branch being merged in must be ahead of the checked out branch. The checked out branch's pointer will just be moved forward to point to the same commit as the other branch.
- the regular type of merge
  - two divergent branches are combined
  - a merge commit is created

WARNING

It's very important to know which branch you're on when you're about to merge branches together. Remember that making a merge makes a commit. If you make a merge on the wrong branch, use this command to undo the merge:

```
git reset --hard HEAD^
```

Undoing changes

- Changing the last commit

```
git commit --amend
```

- Reverting a commit

```
git revert <SHA-of-commit-to-revert>
```

Working with remotes

Difference between git and GitHub

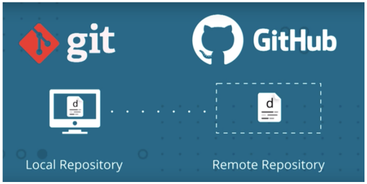
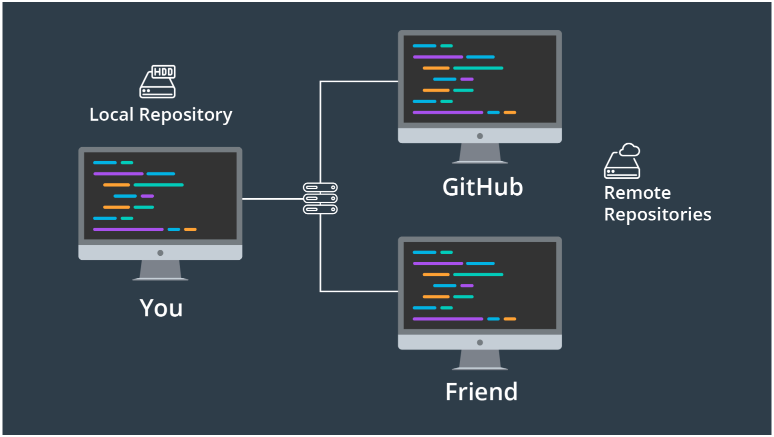


Figure: Git vs GitHub

- Git: tool to manage the repository
  - Git is a version control tool
- GitHub: Hosting version control repositories
  - GitHub is a service to host Git projects

Working with remotes Cont

A local repository is the one that you work on in your local machine. Remote repositories live elsewhere (e.g. a friend's computer, GitHub, etc.)



Commands for remote

- Create & Manage remote repository

```
git remote
```

- To send changes up to the remote

```
git push
```

- Retrieve updates from the remote with merge to local

```
git pull
```

- Retrieve updates from the remote without merge to local

```
git fetch
```

Add remote

- The `git remote` command will let you manage and interact with remote repositories.

```
git remote
```

- Try running this command on the local repository that you haven't shared with anyone yet. What do you get?
- If you haven't configured a remote repository then this command will display nothing
- Try running this command on the cloned repository
- Your repository will automatically have a remote because it was cloned from the repository at the URL you provided
- To see the full path to the remote repository use the `-v` flag

```
git remote -v
```

Hosting on GitHub

- If you don't have an account yet, sign up for one on GitHub Sign Up Page
- The new repository link takes you to the repository creation page. We only need to fill out just one field in this form - the repository name field. Typically you want to use the name of your project as the name of the repository.
- **Don't Initialize with a README** Make sure that you leave the "Initialize this repository with the README" unchecked. We'll be providing our own README, so we don't want GitHub to provide one automatically.
- Also, if we let GitHub auto-generate a new README file for us, then we won't be provided with the setup commands to run in the terminal. It's still possible to get that information, but it will be hidden away
- Now click "Create repository" button to create your remote repository!

Add remote-repo to your local-repo

- Create a connection from your local repository to the remote repository you just created on your GitHub account using

```
git remote add origin <repo-on-GitHub>
```

- To see the details about a connection to a remote.

```
git remote -v
```

Push changes to a remote

- To send local commits to a remote repository you need to use the `git push` command.
- You provide the remote short name and then you supply the name of the branch that contains the commits you want to push:

  `git push <remote-shortname> <branch>`

- Your remote's shortname is `origin` and the commits that you want to push are on the `master` branch.
- Use the following command to send your commits to the remote repository on GitHub:

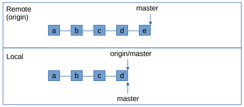
```
git push origin master
```

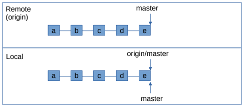
Pull changes from a remote

- To pull changes from remote repo and merge to local repo implicitly

```
git pull origin master
```

- Do `pull` when:
  - Working with a team, and a co-worker has pushed new changes to the remote.
  - Working on the same project but from different computers.
- When git pull is run, the following things happen:
  - The commit(s) on the remote branch are copied to the local repository and the local tracking branch (origin/master) is moved to point to the most recent commit
  - The local tracking branch (origin/master) is merged into the local branch (master)



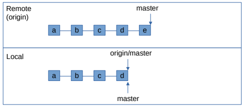


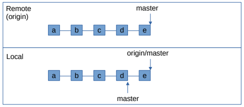
Fetch changes from a remote

- If you don't want to automatically merge the local branch with the tracking branch then you wouldn't use git pull `git pull` you would use a different command `git fetch`.
- To fetch changes from remote repo but merge to local repo explicitly

```
git fetch origin master
```

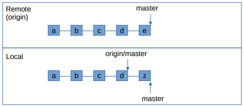
- You might want to do this if there are commits on the repository that you don't have but there are also commits on the local repository that the remote one doesn't have either.
- When git fetch is run, the following things happen:
  - The commit(s) on the remote branch are copied to the local repository and the local tracking branch (origin/master) is moved to point to the most recent commit
  - The local tracking branch (origin/master) is not merged into the local branch (master)



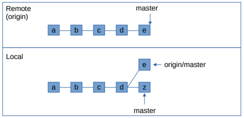


Fetch changes from a remote cont.

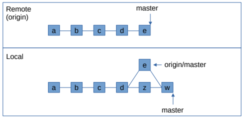
1. One main point when you want to use `git fetch` rather than `git pull` is if your remote branch and your local branch both have changes that neither of the other ones has.



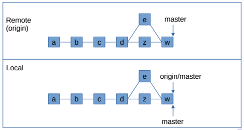
2. In this case, you want to fetch the remote changes to get them in your local branch



3. Then perform a merge manually.



4. Then you can push that new merge commit back to the remote.



Working on another developer repo

- Forking a repo
- Reviewing another developer changes
- Knowing what changes to make

There is no fork command

`There is no fork command!`

- Its provided by the hosting environment.
- In version control terminology if you "fork" a repository that means you duplicate it. Typically you fork a repository that belongs to someone else. So you make an identical copy of their repository and that duplicate copy now belongs to you.
- If a repository doesn't belong to your account, then it means you don't have permission to modify it;
- If you fork the repository to your own account then you will have full control over that repository.

The pull request

Pull request is a request for the source repository to pull in your commits and merge them with their project. To create a pull request, a couple of things need to happen:

1. you must fork the source repository
2. clone your fork down to your machine
3. make some commits
4. push the commits back to your fork
5. create a new pull request and choose the branch that has your new commits

Stay in sync with source project

If you want to keep up-to-date with the Repository, GitHub offers a convenient way to keep track of repositories - it lets you star repositories

- You can go to https://github.com/stars to list out and filter all of the repositories that you have starred.
- If you need to keep up with a project's changes and want to be notified of when things change, GitHub offers a "Watch" feature.

Stay in sync with source project cont.

Add connection to the remote source repository.

```
git remote add upstream <source-repo-on-GitHub>
```

Origin vs Upstream:

- One thing that can be a tiny bit confusing right now is the difference between the origin and upstream. What might be confusing is that origin does not refer to the source repository that we forked from. Instead, it's pointing to our forked repository. So even though it has the word origin is not actually the original repository.
- Remember that the names origin and upstream are just the default or de facto names that are used. If it's clearer for you to name your origin remote mine and the upstream remote source-repo:

```
git remote rename origin mine
git remote rename upstream source-repo
```

Stay in sync with source project cont.

- Get the changes from upstream remote repository

```
git fetch upstream master
```

- Then merge it with local repo

```
git merge upstream/master
```

- Then push it your remote repo account

```
git push origin master
```

- Retrieve updates from the upstream with merge to local repo

```
git pull upstream master
```