WORKSHOPS

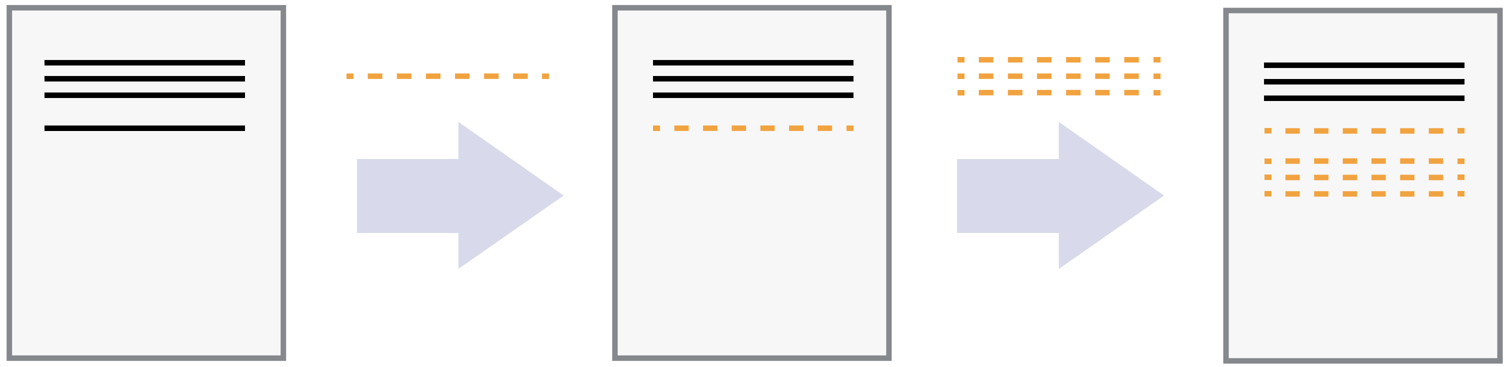
Week 6 –Version control with git

VERSION CONTROL WITH GIT

THE UNIVERSITY *of* EDINBURGH
School of Engineering

WHAT IS VERSION CONTROL?

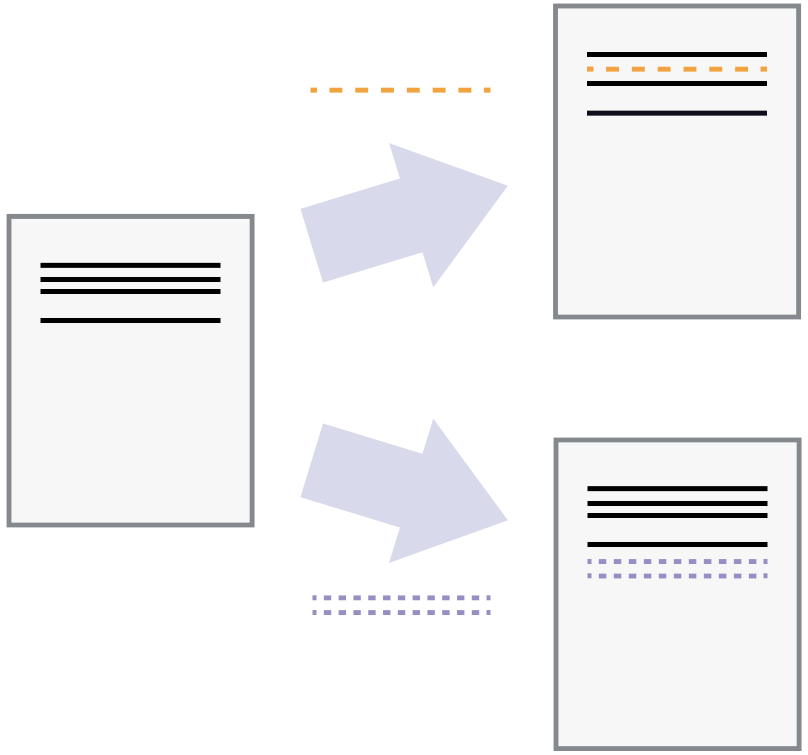
Version control systems start with a base version of the document and then record changes you make each step of the way. You can think of it as a recording of your progress: you can rewind to start at the base document and play back each change you made, eventually arriving at your more recent version.



THE UNIVERSITY *of* EDINBURGH
School of Engineering

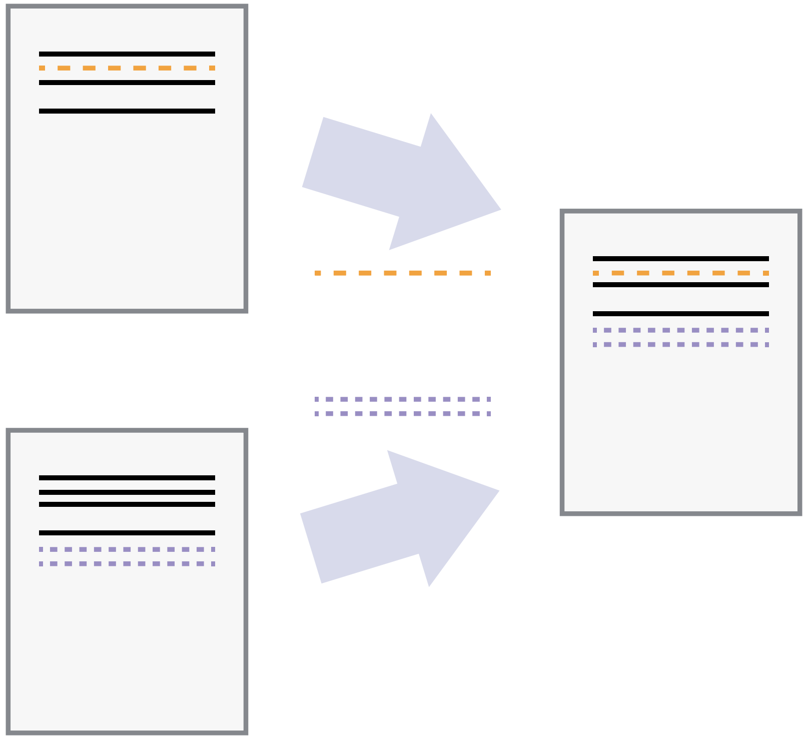
WHAT IS VERSION CONTROL?

Once you think of changes as separate from the document itself, you can then think about "playing back" different sets of changes on the base document, ultimately resulting in different versions of that document. For example, two users can make independent sets of changes on the same document.



WHAT IS VERSION CONTROL?

Unless multiple users make changes to the same section of the document - a conflict - you can incorporate two sets of changes into the same base document.



WHAT IS VERSION CONTROL?

A version control system is a tool that keeps track of these changes for us, effectively creating different versions of our files. It allows us to decide which changes will be made to the next version (each record of these changes is called a commit), and keeps useful metadata about them. The complete history of commits for a particular project and their metadata make up a repository. Repositories can be kept in sync across different computers, facilitating collaboration among different people.

SETTING UP

When we use Git on a new computer for the first time, we need to configure a few things. Below are a few examples of configurations we will set as we get started with Git:

- our name and email address,
- what our preferred text editor is,
- and that we want to use these settings globally (i.e. for every project).

SETTING UP

On a command line, Git commands are written as git verb options, where verb is what we actually want to do and options is additional optional information which may be needed for the verb. So here is how Dracula sets up his new laptop:

In Noteable to get a bash terminal go to new->terminal

Bash

```
$ git config --global user.name "Vlad Dracula"
$ git config --global user.email "vlad@tran.sylvan.ia"
```

Please use your own name and email address instead of Dracula's. This user name and email will be associated with your subsequent Git activity, which means that any changes pushed to GitHub, BitBucket, GitLab or another Git host server after this lesson will include this information.

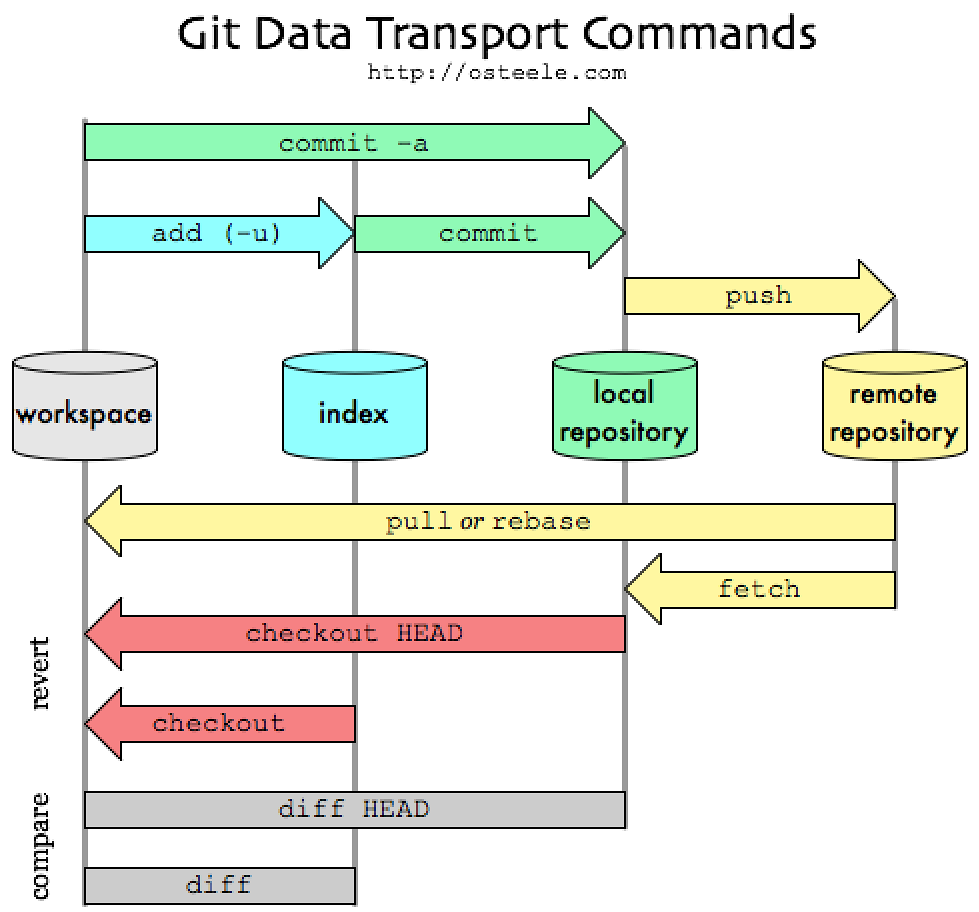
KEY POINTS

Git uses this special subdirectory to store all the information about the project, including all files and sub-directories located within the project's directory. If we ever delete the .git subdirectory, we will lose the project's history.

KEY POINTS

- Version control is like an unlimited ‘undo’.
- Version control also allows many people to work in parallel.
- git init initialises a repository. When we initialise a repository git is keeping track of all the changes.
- Git stores all of its repository data in the .git directory.

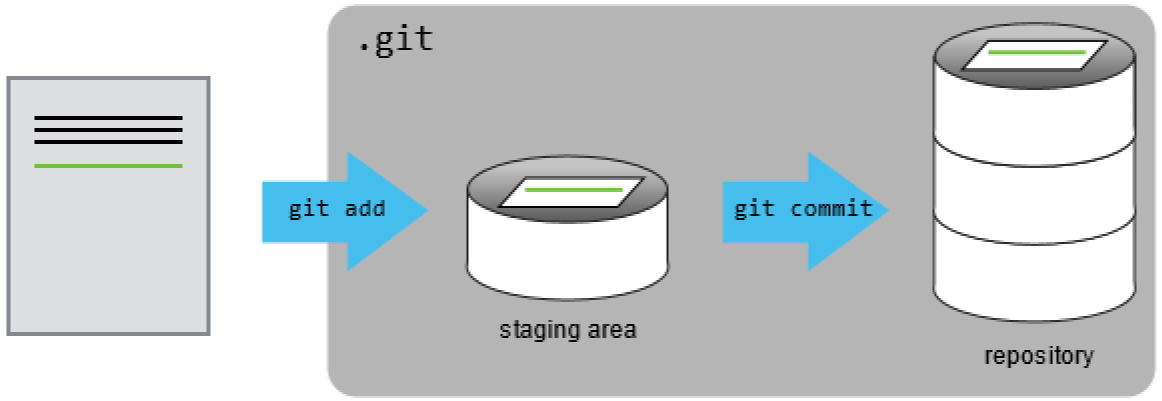
GIT WORKFLOW



Source: Oliver Steele

GIT ADD

When we run git add, git adds the changed file to the staging area. The staging area stores the changed file temporarily. You can stage multiple times and files before committing unless you otherwise specify.



Example:
We made some changes to a file called mars.txt. Those changes were about Mars' climate

```
git add mars.txt
```

THE UNIVERSITY *of* EDINBURGH
School of Engineering

GIT COMMIT

When we run git commit, Git takes everything we have told it to save by using git add and stores a copy permanently inside the special .git directory. This permanent copy is called a commit (or revision) and its short identifier is a series of number and characters.

We use the -m flag (for “message”) to record a short, descriptive, and specific comment that will help us remember later on what we did and why.

Example:

```
git commit -m "Discuss concerns about Mars' climate”
```

THE UNIVERSITY *of* EDINBURGH
School of Engineering

GIT MODIFY-ADD-COMMIT CYCLE

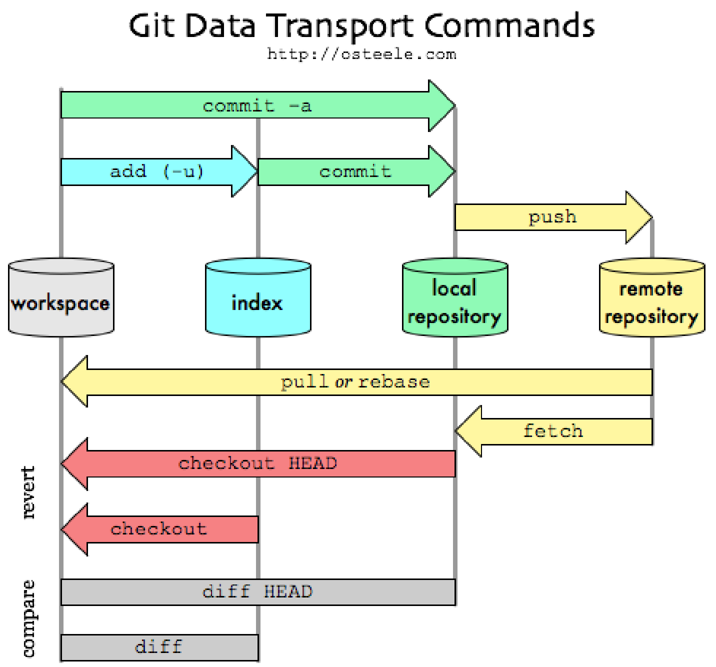
When we want to add changes to our repository, we first need to add the changed files to the staging area (git add) and then commit the staged changes to the repository (git commit)

Files can be stored in a project's working directory (which users see), the staging area (where the next commit is being built up) and the local repository (where commits are permanently recorded).

OTHER IMPORTANT COMMANDS

Git log:	lists all commits made to a repository in reverse chronological order.
Git diff:	displays differences between commits.
Git status:	shows the status of a repository.
Git checkout:	git checkout recovers old versions of files.

PUSHING CHANGES



We often want to share our commits with others via a remote repo (for example on Github).

git push will attempt to add your commits to the remote repo, so long as there are no conflicts

Example:

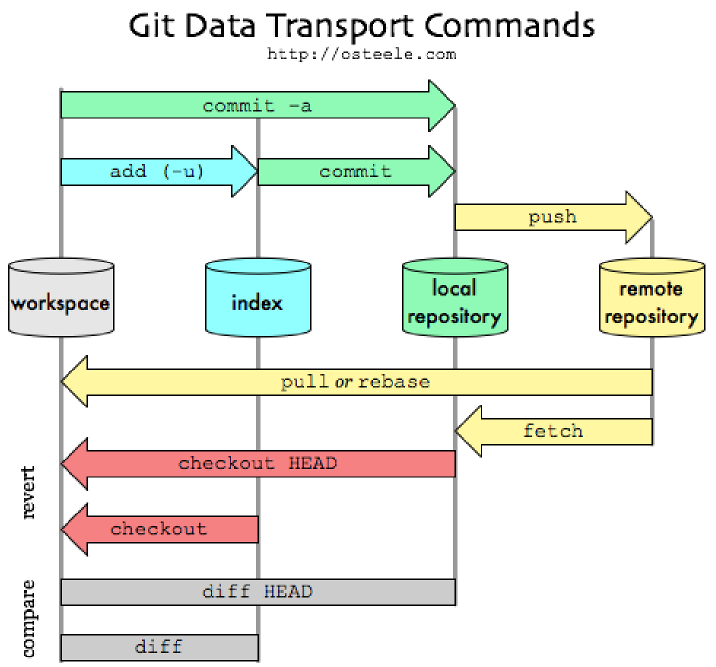
```
git push
```

THE UNIVERSITY *of* EDINBURGH
School of Engineering

PULLING CHANGES

The opposite of **git push** is **git pull** where we want to get commits from others via the remote repo.

git push will attempt to add remote commits to the local repo, so long as there are no conflicts

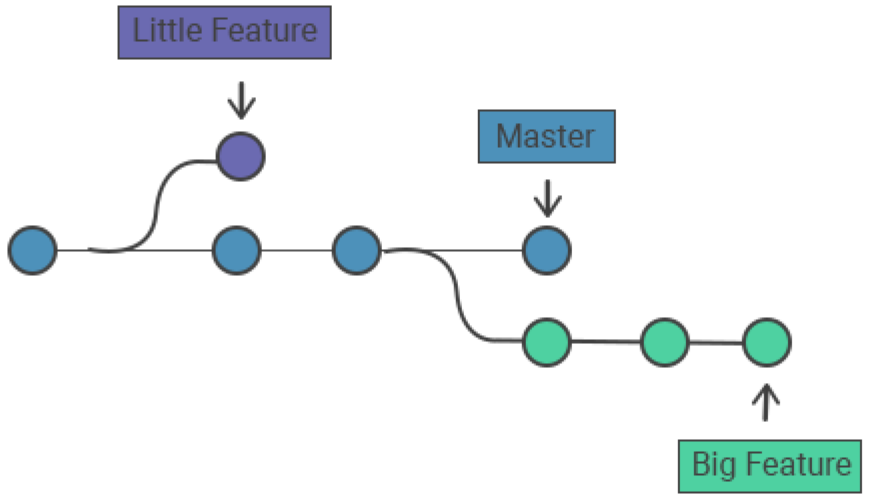


Example:

```
git pull
```

BRANCHES

Branches represent an isolated line of development. They are accepted as a way to request a new working directory, staging area and project history. Developing isolated lines of development for two features in branches will enable to operate on them in parallel and make the master branch free from questionable code.



LINKEDIN LEARN GIT COURSE

Git essential training

GIT CHEAT SHEETS FOR QUICK REFERENCE

- Printable Git cheatsheets in several languages are available here (English version). More material is available from the GitHub training website.
- An interactive one-page visualisation about the relationships between workspace, staging area, local repository, upstream repository, and the commands associated with each (with explanations).
- Both resources are also available in other languages (e.g. Spanish, French, and more).
- Open Scientific Code using Git and GitHub - A collection of explanations and short practical exercises to help researchers learn more about version control and open source software.
- GIT cheat sheet

REFERENCES

1. Rosett C.M., Hagerty A. (2021) Data Wrangling. In: Introducing HR Analytics with Machine Learning. Springer, Cham. https://doi.org/10.1007/978-3-030-67626-1_13
2. Stefanski, R., Sinha, V. and Poddar, A., 2022. *Data Wrangling in 6 Steps: An Analyst's Guide For Creating Useful Data*. [online] Learn | Hevo. Available at: https://hevodata.com/learn/data-wrangling/#s2
3. Tripathi, S., Muhr, D., Brunner, M., Jodlbauer, H., Dehmer, M. and Emmert-Streib, F., 2021. Ensuring the Robustness and Reliability of Data-Driven Knowledge Discovery Models in Production and Manufacturing. *Frontiers in Artificial Intelligence*, 4.

USING GIT

1. Before starting:
   a) Get yourself a Github account and email it to a.sherlock@strath.ac.uk
   b) Accept my invite to collaborate on the drewsherlock/dsim_workshops repo

2. Set up a ‘Personal Access Token’ from Github:
   a) Account->Settings-> Developer Settings (bottom left)->Personal Access Tokens
   b) Chose a suitable expiration date and select ‘repo’ for the scope.
   c) I suggest pasting the token into a text file for later

3. Open a terminal on Notable (New->Terminal)
   a) Try some Linux commands
      - ***ls –al*** (lists all files in folder)
      - ***mkdir my_new_dir***
      - ***cd my_new_dir*** (change to new directory)
      - ***cd ..*** (change up a directory)
   b) Return to your home directory with ***cd ~***

USING GIT

4. Clone the dsim_workshops repo with command:

   git clone https://github.com/drewsherlock/dsim_workshops.git

5. Change directory into the Workshops repo:

   cd dsim_workshops

6. Try the following:

   git status get info on status of repo
   git log -5 *see info on last 5 commits*
   Open the Git_Practise_Notebook.ipynb, follow instructions and save it
   git status should now see a notebook file with changes
   git checkout –b <your_github_username> creates a branch with your name (branch name could be anything but your username is convenient here)
   git add <changed notebook filename> adds changed file to staging area

THE UNIVERSITY *of* EDINBURGH
School of Engineering

USING GIT

6. Try the following (cont.):

git commit –m "My first commit!"	create a commit (better to use a more informative commit message). You'll likely get an error.
git config --global user.email "you@email.com"	add your email address
git config --global user.name "your name"	add your name
git commit –m "My first commit!"	try again!
git log -3	should see your commit
git push	try to push to remote repo. Likely error because it doesn't know what remote branch to push to.

git push --set-upstream origin <branch_name> this tells git where to push this branch to and attempts to push. You will need your Github username and use the personal access token you generated before as the password.

USING GIT

6. Try the following (cont.):

   git branch -a shows all available branches. When they've got this far you should see your buddy's branch

   git switch <buddys_branch> switch to buddy's branch

   git pull

   Open the notebook and add comments back.

   Add then commit the changed file and push up to Github. Your buddy should be able to see your commit on their branch (and you should see their commit on your branch). You can both then pull those commits pull and review.