

Demystifying Git

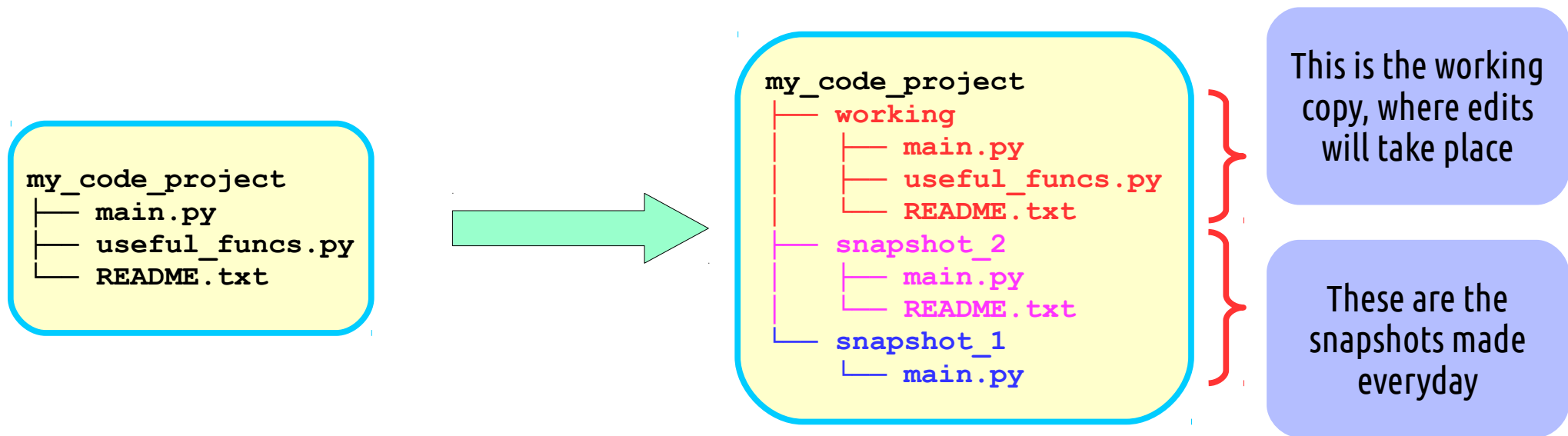
Mark Slater

mslater<at>cern.ch, Physics West 317

UNIVERSITY OF
BIRMINGHAM

Developing a VCS: Saving a Copy Everyday

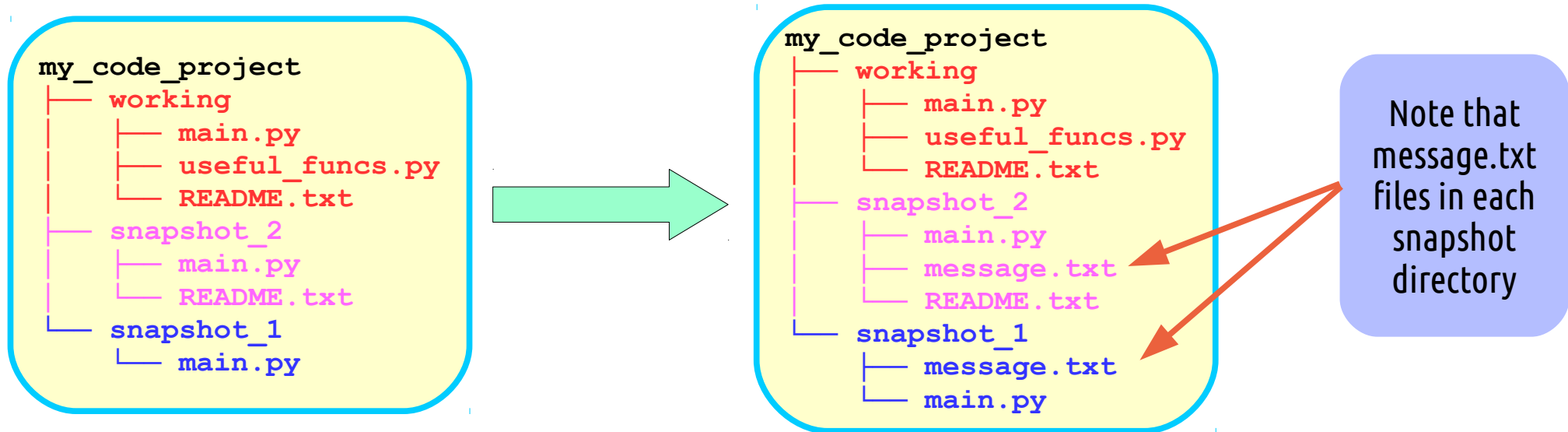
- To try to help explain what Git does, let's go through the steps of essentially coming up with our own VCS
- The most simple VCS is essentially just taking copies (or 'snapshots') of all the project's files and putting them in a separate directory



- This already ticks several of the boxes we wanted for VCS – reproducibility, backup, etc. and at its core, this is all Git is doing!

Developing a VCS: What did I do again?

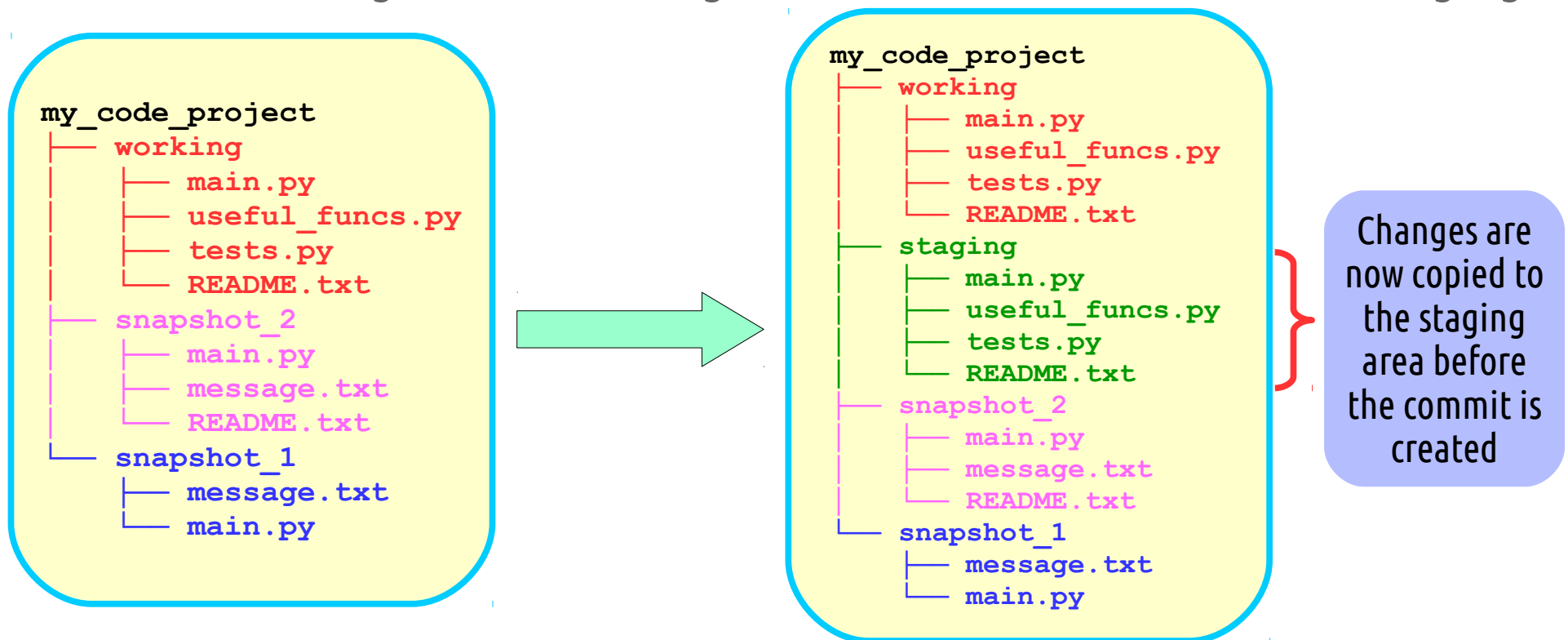
- A significant thing that isn't present when just copying a project's directory is knowing what you did and why
- To get around this, let's add a text file in each snapshot (let's call it a commit from now on) that includes a short message about what has changed since the last commit with the author and date/time info of the commit



- We now have a functional VCS! However, it's not very efficient and is a bit cumbersome to use.

Developing a VCS: One thing at a time

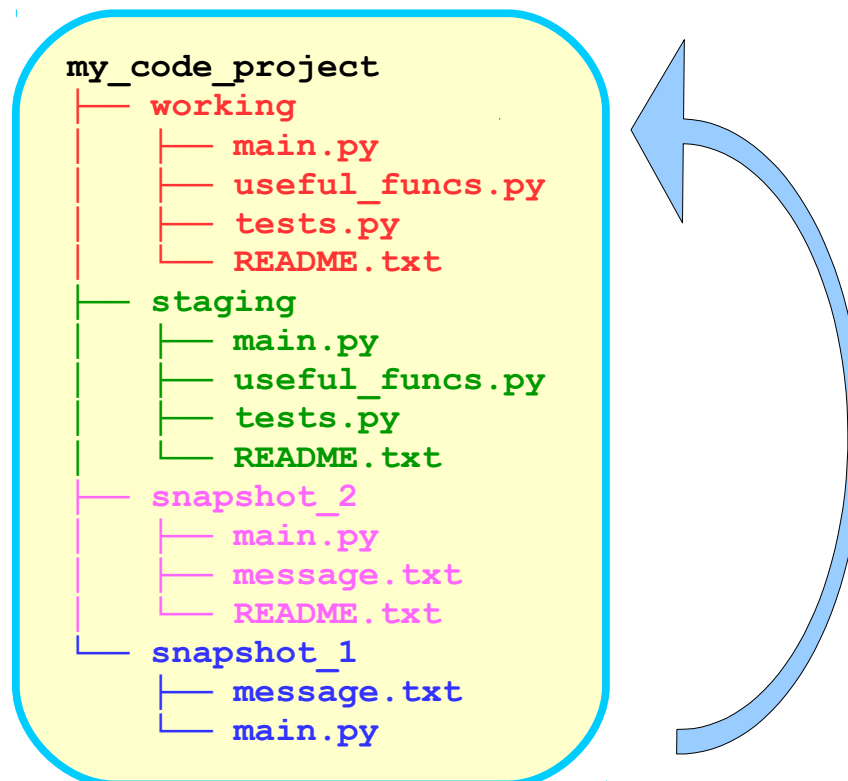
- At present, each commit is just a copy of the working directory every day, no matter what has been done
- But what if you get to the end of the day and have 2 or 3 completely different changes that should go in different commits? Have a staging area!



- You can now choose which changes to add to a particular commit before actually committing them

Developing a VCS: Oops - I caused massive breakage

- What happens if you find that 2 commits ago, you managed to break a crucial feature?
- What we need to do is copy the appropriate file from the appropriate commit to our working area ('checkout' the file) and then perform a commit



Developing a VCS: Making a right hash of things

- As you can probably tell, the names for commits are prone to error and may not work if someone else puts in a commit – how do we get around this?
- Hashing is a very good way to create unique names for things easily as:
 - It will produce an (almost) unique fixed length string for any input
 - Small variations in the data will produce very different hashes
 - It is computationally very quick
- So can we use the only unique file in each commit ('message.txt') to generate a hash and use that as the directory name for the commit?

```
my_code_project
├── working
│   [ 4 files ]
├── staging
│   [ 4 files ]
├── 99b52473039acea4427e13e42b96c78776e2baf5 (snapshot_4)
│   [ 5 files ]
├── d9accd0a27c78b4333d70ee1a9d7dca0bcc3e682 (snapshot_3)
│   [ 5 files ]
└── 00d03e9d1bf4ebaea380da3c62e9226189e39ff4 (snapshot_2)
    [ 4 files ]
```

Note that this is the source of all the strings of hexadecimal numbers you will deal with in git!

- In theory, yes, but now we don't know what order the commits were made in...

Developing a VCS: Linked in

- In order to restore the history, we need each commit message to know what it's parent(s) was
- The hash of the parent can simply be added in a 'Parent' field in the commit message when committing
- You can then reconstruct the history of your project from these commit messages but you still get to use the hashed commit names

```
my_code_project
├─ working
│  [ 4 files ]
├─ staging
│  [ 4 files ]
├─ c20351... (snapshot_4)
│  [ 5 files ]
├─ bee09a... (snapshot_3)
│  [ 5 files ]
└─ 905376... (snapshot_2)
   [ 4 files ]
```

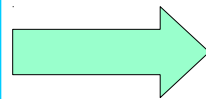
Message.txt contains -
Parent: bee09a...

- Note that, because the message.txt has changed for each commit, the hash has also changed
- Also, I will start abbreviating the hashes as git does

Developing a VCS: Making an even bigger hash of things

- As you make commits, you will notice you get a copy of **every** file – this means your project directory growing continually due to duplicates
- This is where hashes come in again – if you **create a hash from the contents of a file** during a commit and it is the same as another one, these files are the same
- You can then just save a reference rather than an additional copy of the file

```
my_code_project
├── working
│   [ 4 files ]
├── staging
│   [ 4 files ]
├── c20351... (snapshot_4)
│   [ 5 files ]
├── bee09a... (snapshot_3)
│   [ 5 files ]
└── 905376... (snapshot_2)
    [ 4 files ]
```



```
my_code_project
├── working
│   [ 4 files ]
├── staging
│   [ 4 files ]
├── repo
│   ├── objects
│   │   ├── 18e92b (main.py)
│   │   ├── 27e85e (useful_funcs.py)
│   │   ├── 47eef8 (README.txt)
│   │   └── 4e3c43 (tests.py)
│   ├── c20351... (snapshot_4)
│   │   ├── directory_listing.txt
│   │   └── message.txt
│   ├── bee09a... (snapshot_3)
│   │   ├── directory_listing.txt
│   │   └── message.txt
│   └── 905376... (snapshot_2)
│       ├── directory_listing.txt
│       └── message.txt
```

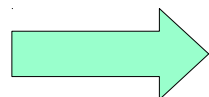
Files renamed
to their
computed hash
value

Directory Listing files contain
things such as
18e92b... main.py
27e85e... useful_funcs.py

Developing a VCS: Cleaning up

- You can actually take the storing of hashed files even further by hashing the contents of 'message.txt' and 'directory_listing.txt' files and moving to the 'objects' directory as well
- You need to add a reference to the correct 'directory_listing.txt' file in an additional field to 'message.txt' and also an additional file to point to the last commit

```
my_code_project
├── working
│   [ 4 files ]
├── staging
│   [ 4 files ]
└── repo
    ├── objects
    │   ├── 18e92b (main.py)
    │   ├── 27e85e (useful_funcs.py)
    │   ├── 47eef8 (README.txt)
    │   └── 4e3c43 (tests.py)
    ├── c20351... (snapshot_4)
    │   ├── directory_listing.txt
    │   └── message.txt
    ├── 9920ff... (snapshot_3_jane)
    │   ├── directory_listing.txt
    │   └── message.txt
    ├── bee09a... (snapshot_3)
    │   ├── directory_listing.txt
    │   └── message.txt
    └── 905376... (snapshot_2)
        ├── directory_listing.txt
        └── message.txt
```



```
my_code_project
├── working
│   [ 4 files ]
├── staging
│   [ 4 files ]
└── repo
    ├── my_bookmark
    │   └── 47eef8
    ├── objects
    │   ├── 18e92b
    │   ├── 27e85e
    │   ├── 47eef8
    │   ├── 4e3c43
    │   └── ...
```

The my_bookmark contains the hash of the latest commit (message.txt file) which in turn, knows about its parent and the files it contains

All content files, message files and directory listing files are now renamed with the hash of their contents

Developing a VCS: What we've learned

- This is now a fairly close approximation to what git does
- Most importantly though, hopefully this will help you understand some of the terminology git uses and what it's trying to do:
 - ➔ Repository – The folder with all the files associated with the project and git are located
 - ➔ Index – What git calls the 'staging area'
 - ➔ Commit – creating a copy of the index, adding a message and updating the hash pointers
 - ➔ Hash – Used to create unique filenames based on the file contents
 - ➔ HEAD – the hash that points to the last commit of the current branch you're working on, used to compare the index with when committing.

Good Git Practise

- When working with git (and any VCS actually), there are few general rules:
 1. **Only include source files**
 - You shouldn't add anything that can be created from the source files (e.g. *.pyc, *.o, etc.)
 2. **Write good commit messages**
 - The commit messages can be long so don't just put 'made some changes'
 3. **Commits should be related**
 - Only include changes that are related in any one commit
 4. **Keep commits small**
 - Large changes in single commits can be confusing and difficult to solve conflicts
 5. **Only commit completed work**
 - Git isn't a backup system – only commit things that are complete and tested