Source code management

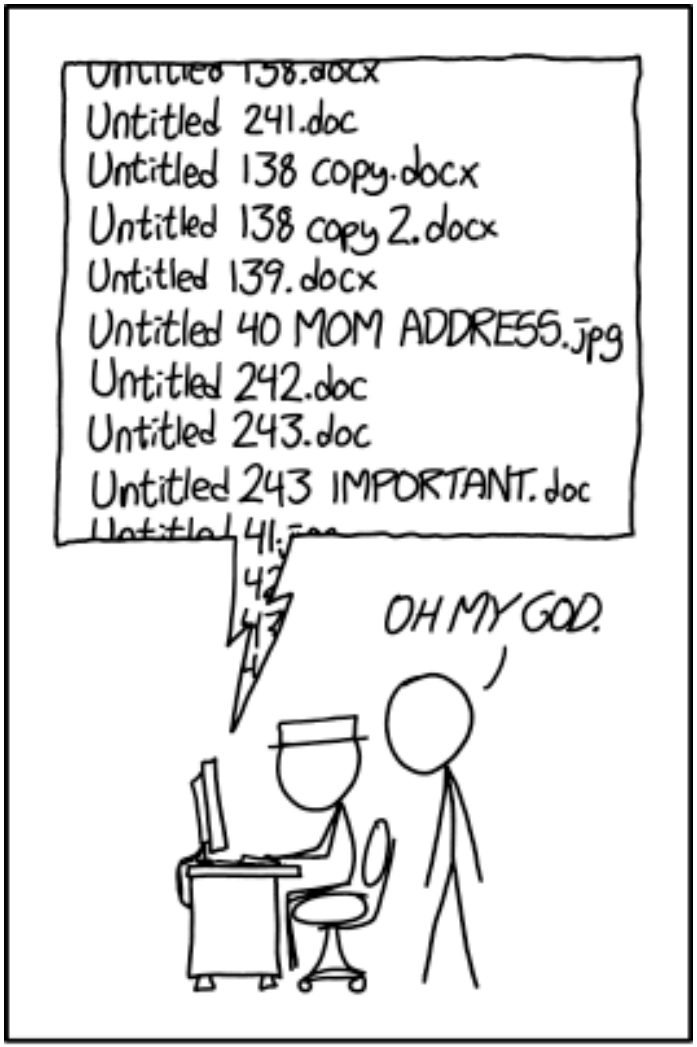
Get yourself organised

Version 1

- You work hard on something
- You keep saving over it
- You realise you want the older version back
- There is only one file
- No undo, no choices

Version 2

- You work hard on something
- You save another copy of it
- You realise that this new version is not working out
- You keep making copies of the file and reassure yourself it is a “backup”



Version 3

- You work on something, but it is not hard work…

- It is hard because
  - you are dealing with several copies
  - there are 0 or more changes between successive copies
  - you don’t understand all the differences

Large programs

- Teams of programmers
- Same files being worked on at the same time
- Have to synchronise somehow
- Huge code bases
  - Boeing 787 ~14 million lines of code
  - Linux kernel ~19.1 million lines of code (2014)

It’s not the size that counts!

The Inside Story of Mt. Gox, Bitcoin’s $460 Million Disaster

“Mt. Gox, he says, didn’t use any type of version control software — a standard tool in any professional software development environment.”

“The source code was a complete mess,”

Bitcoin core 12K LOC -> 89K LOC

Source Code Control Systems

- Tools called *Source Code Control Systems* are used to coordinate access to the program files
- large systems require *version control*:
  - maintaining several versions of a program simultaneously (eg normal and Pro versions)
  - ability to revert to an older version
  - maintain current released product version while working on the next version

Source Code Control

- Many source code control systems available: some commercial and some open source
- commonly used open source systems include
  - cvs
  - Subversion or “svn”
  - git
  - Mercurial or “hg”
- an essential tool for managing **any** software development project

Source Code Control

- some source code control systems operate a *checkin/ checkout* system
  - a programmer checks out the part of the code they are working on
  - when finished they check the modified code back in
  - the system will warn them if any conflicting changes have been made by another programmer in the mean time
  - check in requires the programmer to document the changes
- source code control systems can be distributed and allow you to check out/in your code via the net

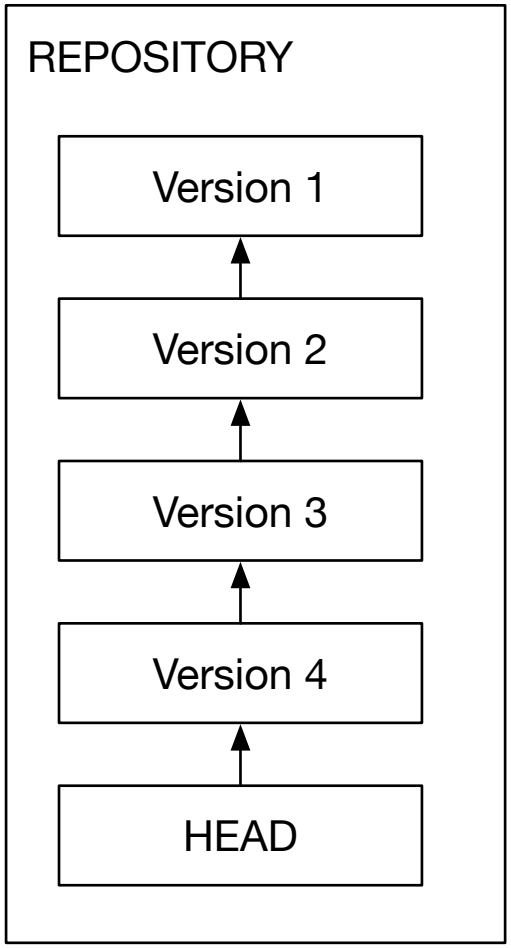
Source Code Management

- other source code control systems use a *clone/push/pull* system
  - a programmer initially “clones” the code repository
  - when finished making modifying the code they “push” their changes back into the repository
  - Before doing new work on the local copy of the code they “pull” any changes that have been made to the repository
- Mercurial and Git are examples of this type of system
- It allows a structure without a central repository like SVN

git

- Repository:
  - A *remote* place where all the information is stored
  - Can be sitting on a server, or even on your computer
  - Setup by owner
  - Initially empty (version 1)

```
$ mkdir reponame.git
$ cd reponame.git
$ git --bare init
```



Git Working directory

On your computer, you request a local copy of what is in the repository "clone"

```
$ git clone reponame.git
$ cd reponame
```

You can make changes to files and folders already there, or add new files/folders

```
$ echo "Hello" > README
$ git add README
```

Git Working directory

If you made changes and are happy about your state

- You can save your progress by performing a commit to the local copy, and/or

```
$ git commit
```

If you are super happy

- you can save your progress and push the changes with the remote repository (synchronise)

```
$ git commit && git push origin master
```

Git Working directory

If you made changes and are not happy about your current state

- Save what you have done, in a stash, and go back to a “previous local copy”

  ```
  $ git stash
  ```

  - Go back to the saved stash

  ```
  $ git stash pop
  ```

- go back to last commit

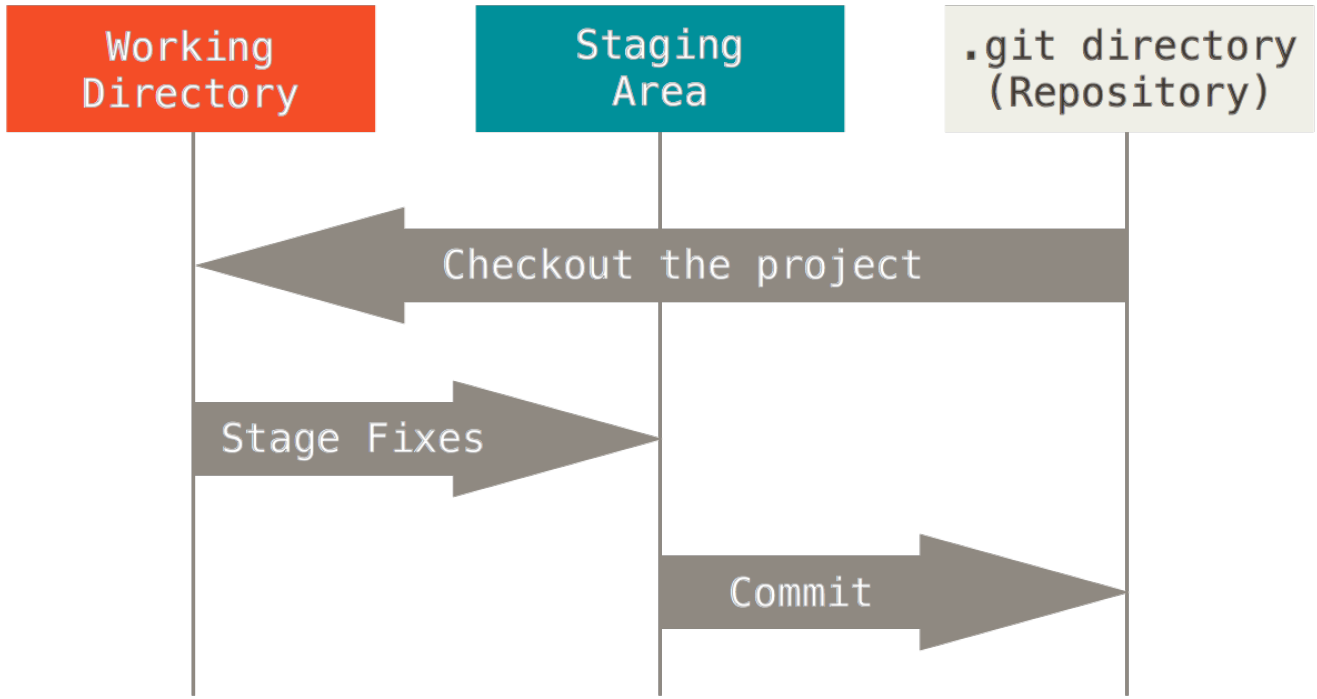
  ```
  $ git revert <version>
  ```

- Delete everything new reset all changes and go back to “last remote copy”

  ```
  $ git reset --hard HEAD
  ```

Git: Local Repository

- *commit* your changes
- *checkout* a previous commit (safe navigation)
- *revert* to a previous commit (undo last, but keep)

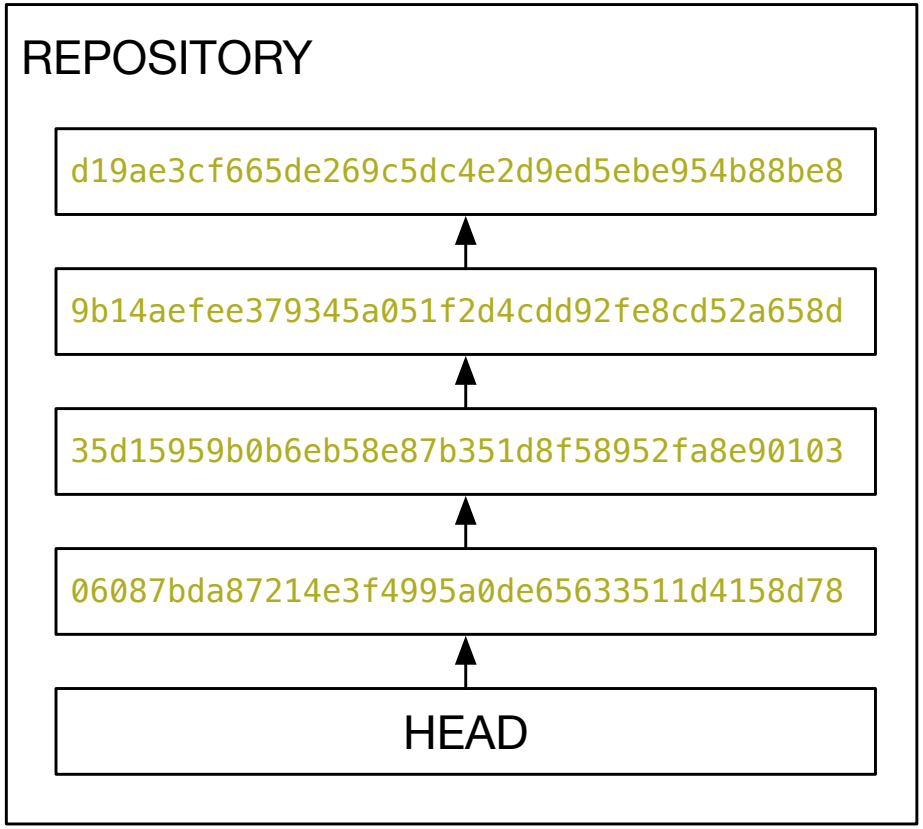


Git: version is a hash value

- Each commit is unique data
- The *version* is described as a SHA1 hash value

```
$ git show
$ git log

$ git checkout
```



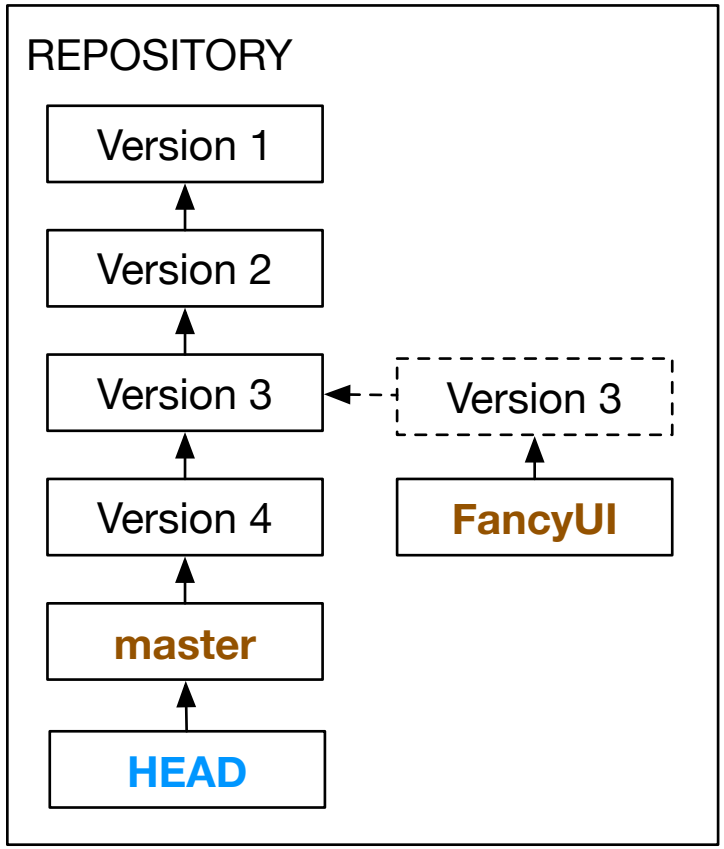
Git: Repository is a tree

- Branches provide a isolated development env
- Maintain stable branch for production
- Anything else…branch:

```
$git branch FancyUI
```

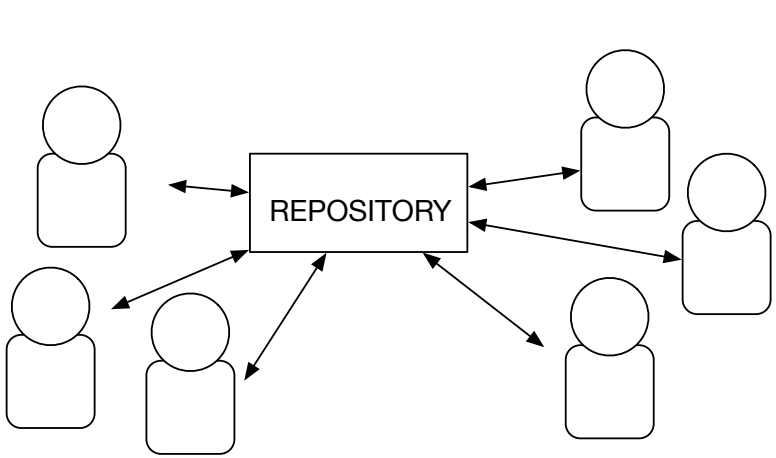
- Switching branches

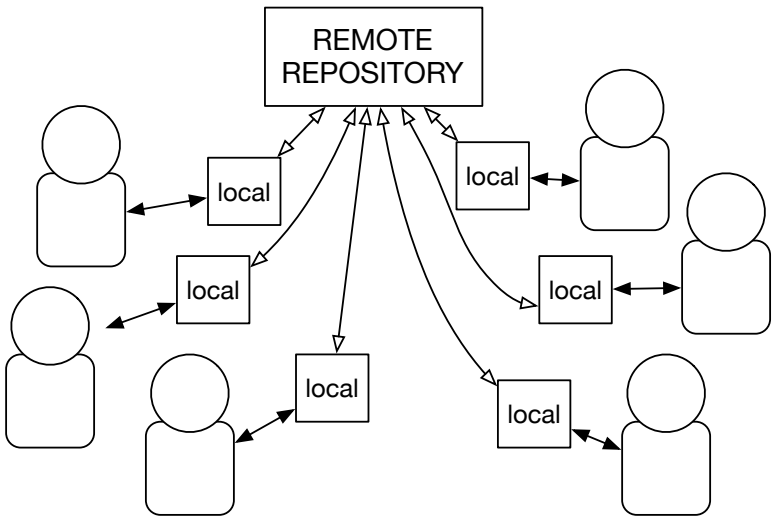
```
$git checkout FancyUI
$git checkout master
```



git

- Distributed nature
  - make a copy of an existing *repository* (*clone*)
  - *push* your changes to another respository
  - *pull* changes from another repository
  - Many configurations possible





Git: Merging changes

- Git will merge changes in files
  - If it knows how
  - If they don't conflict
- Conflicts are reported and can be fixed
- View the diff info, find the file:
  - Merge changes manually, or
  - Pick one file over another

Without this, chaos!

Git: proviso

- It is not the only, nor the best, but very versatile
- Understanding a tool like git is difficult when you haven't been hurt by older Version Control Systems
- The Pro Git book is online and free
  - http://git-scm.com/book/en/v2
- Recommended reference
  - man pages: git help <X>
  - http://gitref.org/basic
  - https://www.atlassian.com/git/tutorials
  - http://try.github.com

Summary

- source code control is essential for large projects with many files and many programmers
- there are many such systems
- usually based on the repository model and checkout/modify/checkin development cycle

- References
  - http://www.wired.com/2014/03/bitcoin-exchange
  - XKCD: “Documents” https://xkcd.com/1459