Fusemachines
AI School

Version Control

Traditional Version Management


newcode.py


newcode1.py


newcode2.py


newcode3.py


newcode4.py

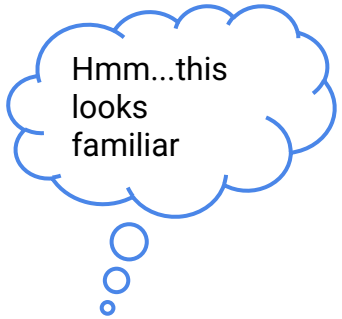

newcode5.py


newcode6.py


newcode7.py


newcodefinal.py


newcodefinal corrected.py



VERSION CONTROL

1 VERSION CONTROL SYSTEM (VCS)

2 GIT WORKFLOW

Version Control

- Tracks changes (& why you made those changes)

Version 2

On January 7, you updated this line on file A

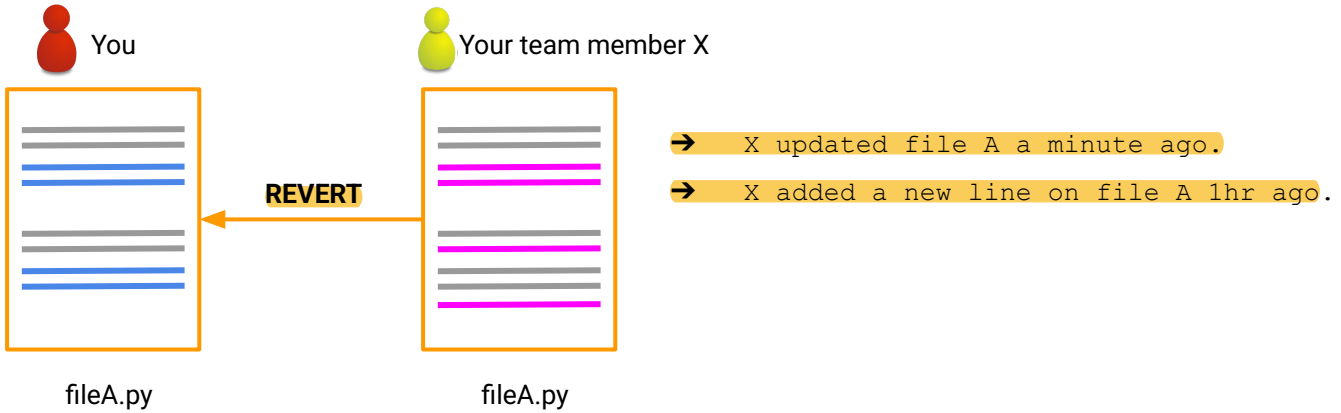
On February 10, you added file B

Version Control

- Compare changes

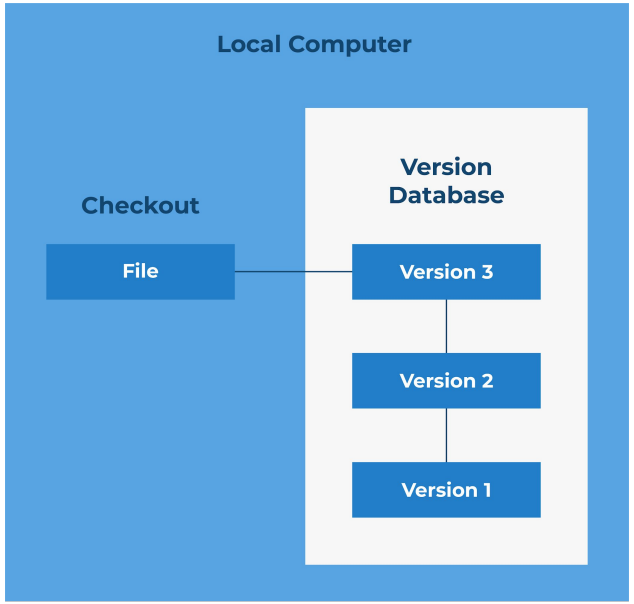
Version Control

- Source Code Management (SCM)
- Collaborative development



Types of Version Control Systems

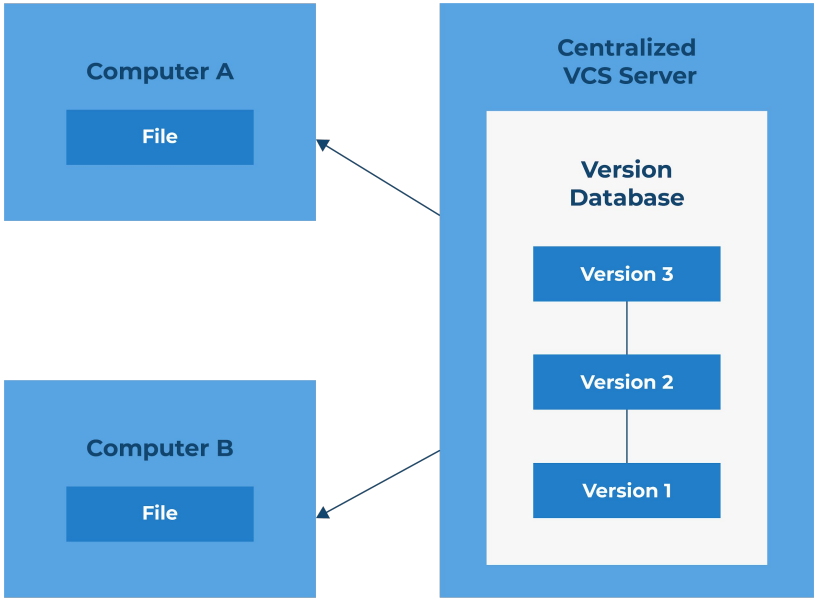
Local Versioning Control System (LVCS)



Local Versioning Control System (LVCS), as Revision Control System (RCS)

Image source: https://git-scm.com/

Centralised Versioning Control System (LVCS)



Centralized Versioning Control System(CVCS)

eg: Tortoise SVN

Image source: https://git-scm.com/

Distributed Versioning Control System (DVCS)

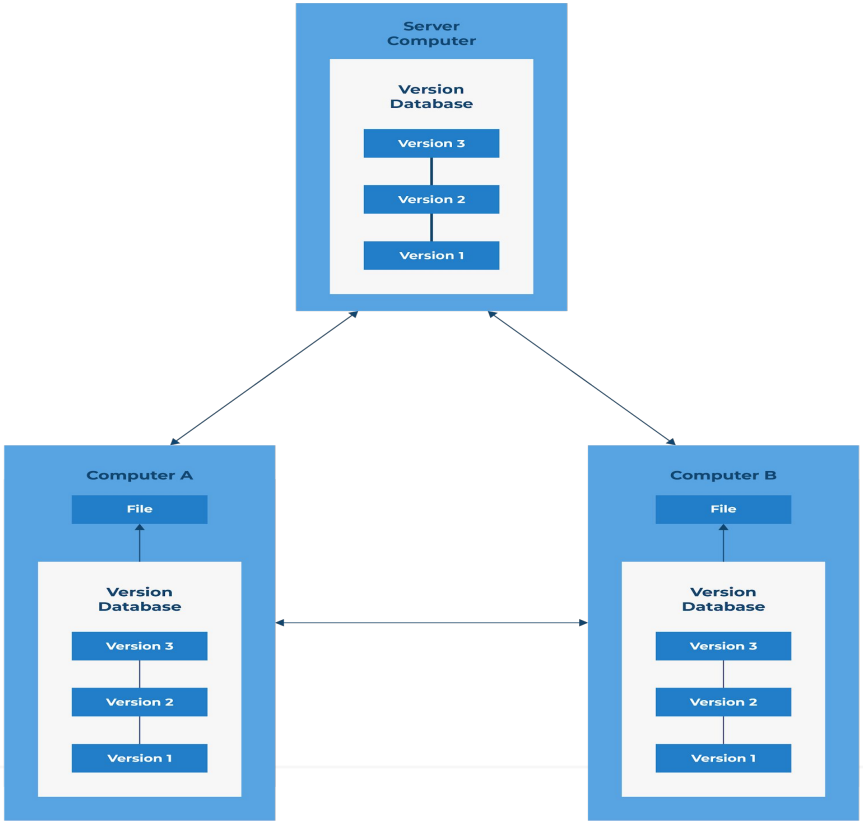


Image source: https://git-scm.com/

GIT

Git is the de facto standard for version control

source: https://git-scm.com/

GIT DATA MODEL

Snapshots

Git models the history of a collection of files and folders within some top-level directory as a series of **snapshots**.

In Git terminology,

1. A file is called a "**blob**"
2. A directory is called a "**tree**"

A snapshot is the top-level tree that is being tracked.

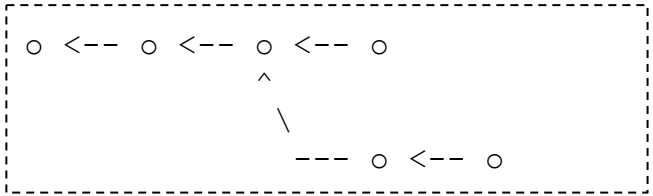
GIT DATA MODEL

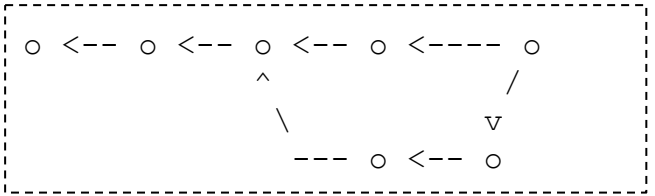
Snapshots

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

GIT DATA MODEL

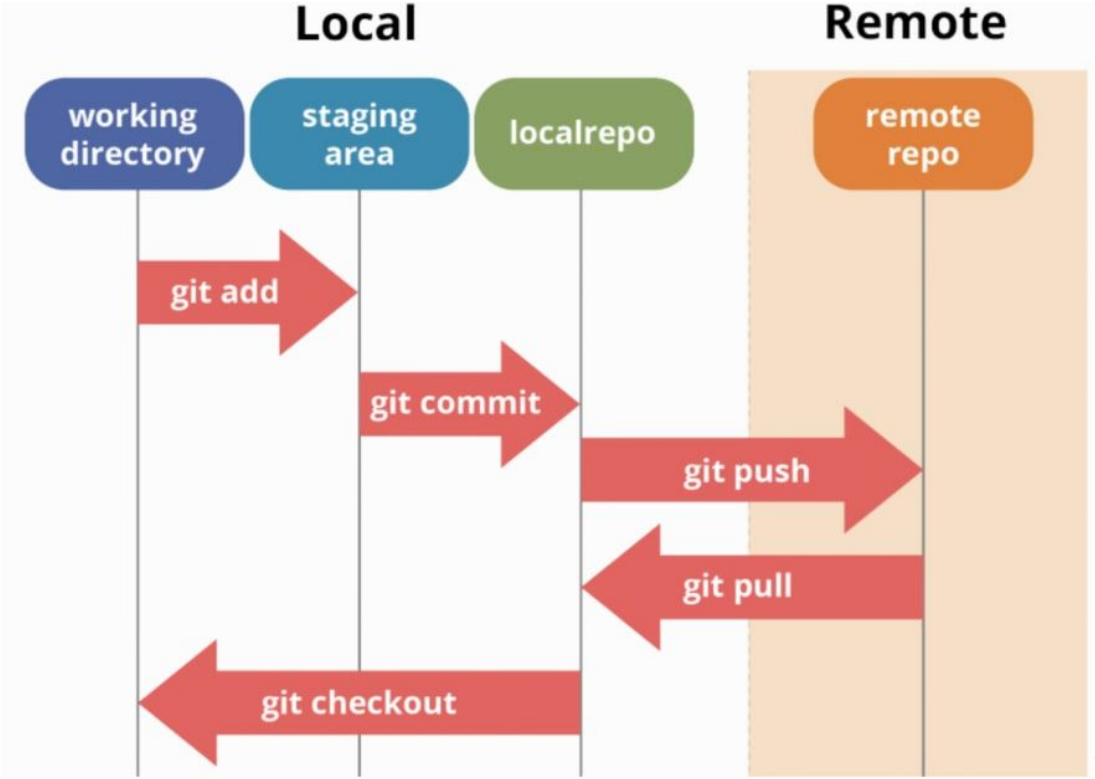
Modeling history: relating snapshots





- In Git, a history is a directed acyclic graph (DAG) of snapshots.
- Git calls these snapshots “**commit**”s.
- Commits in Git are immutable.
- all snapshots can be identified by their SHA-1 hash

GIT Workflow



Committed:
Data is safely stored in local database

Modified:
You have changed the file but not added it to the database

Staged:
You have marked a modified file in its current version to go into your next commit snapshot.

Task 1: Git Basics

1. Initialize a git repository and track changes in “hello.txt” file.

Commands

```
> git init
> echo “Hello World” > hello.txt
> git status
> git add hello.txt
> git status
> git commit -m “ initial commit”
> echo “Another line” >> hello.txt
> git status
> git diff
> git add .
> git commit -m “Add another line”
> git log
> git log --graph --decorate
> git diff <hash_code>
```

Task 2: Branching and Merging

1. Create a python file “animals.py” and implement feature of dog and cat in different branches. Merge the two branches and resolve and merge conflicts.

Commands

```
> touch animals.py
> code animals.py
> write master code
> git add animals.py
> git commit -m “ initial commit”
> git branch cat
> git checkout cat
> code animals.py
> add cat code
> git add animals.py
> git commit -m “Added cat functionality”
> git checkout master
> git log --graph --decorate --oneline
> git checkout -b dog
> code animals.py
> write dog code
> git commit -a -m “Added dog functionality”
> git checkout master
> git merge cat
> git merge dog
> Resolve merge conflict
> git merge --continue
```

master

```
import sys

def default():
    print("Hello World")

def main():
    default()

if   name _ == '__main__':
    main()
```

cat

```
import sys

def default():
    print("Hello World")

def cat():
    print('Meow!')

def main():
    if sys.argv[1] == 'cat':
        cat()
    else:
        default()

if   name _ == '__main__':
    main()
```

dog

```
import sys

def default():
    print("Hello World")

def dog():
    print('Bark!')

def main():
    if sys.argv[1] == 'dog':
        dog()
    else:
        default()

if   name _ == '__main__':
    main()
```

Task 3: Remote Repository

1. Create a github account and push animals.py.

Commands

```
> git remote origin <url> <dirname>
> git push origin master
```

Git command-line interface

- Basics

  `git help <command>`: get help for a git command

  `git init`: creates a new git repo, with data stored in the `.git` directory

  `git status`: tells you what's going on

  `git add <filename>`: adds files to staging area

  `git commit`: creates a new commit

  `git log`: shows a flattened log of history

  `git log --all --graph --decorate`: visualizes history as a DAG

  `git diff <filename>`: show differences since the last commit

  `git diff <revision> <filename>`: shows differences in a file between snapshots

  `git checkout <revision>`: updates HEAD and current branch

Git command-line interface

Branching and merging

`git branch`: shows branches

`git branch <name>`: creates a branch

`git checkout -b <name>`: creates a branch and switches to it

same as `git branch <name>; git checkout <name>`

`git merge <revision>`: merges into current branch

`git mergetool`: use a fancy tool to help resolve merge conflicts

`git rebase`: rebase set of patches onto a new base

Git command-line interface

Remotes

`git remote`: list remotes

`git remote add <name> <url>`: add a remote

`git push <remote> <local branch>:<remote branch>`: send objects to remote, and update remote reference

`git branch --set-upstream-to=<remote>/<remote branch>`: set up correspondence between local and remote branch

`git fetch`: retrieve objects/references from a remote

`git pull`: same as `git fetch; git merge`

`git clone`: download repository from rem

Git command-line interface

Undo

`git commit --amend`: edit a commit's contents/message

`git reset HEAD <file>`: unstage a file

`git checkout -- <file>`: discard changes

Git command-line interface

Advanced Git

`git config`: Git is highly customizable

`git clone --shallow`: clone without entire version history

`git add -p`: interactive staging

`git rebase -i`: interactive rebasing

`git blame`: show who last edited which line

`git stash`: temporarily remove modifications to working directory

`git bisect`: binary search history (e.g. for regressions)

`.gitignore`: specify intentionally untracked files to ignore

Incase something goes wrong!

https://ohshitgit.com/