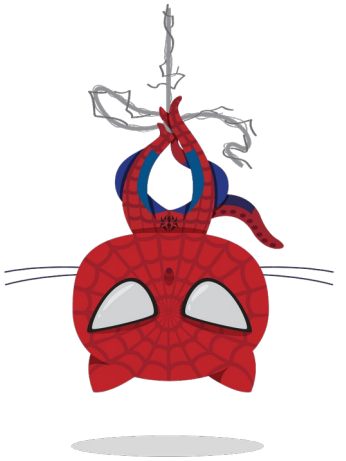
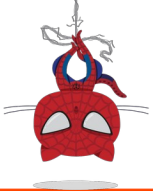


Version Control

Day 1

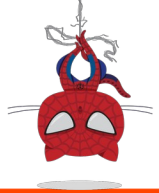
Day 1 Contents



- Introduction to Version Control Systems.
- Centralized Version Control System.
- Distributed Version Control System.
- GIT History.
- Get Started.
- Staging & Remotes & Cloning.

git

What is VCS?

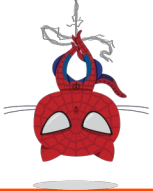


Version Control = Revision Control = Source Control.

- **Version control systems are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database.**
- **If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.**

git

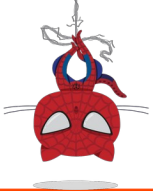
Why do we need VCS?



- You might used your own version control, suppose you are working with a document that needs to be changed periodically, like your CV. So, you may created something like:
  - MyCV_V1
  - MyCV_V2

git

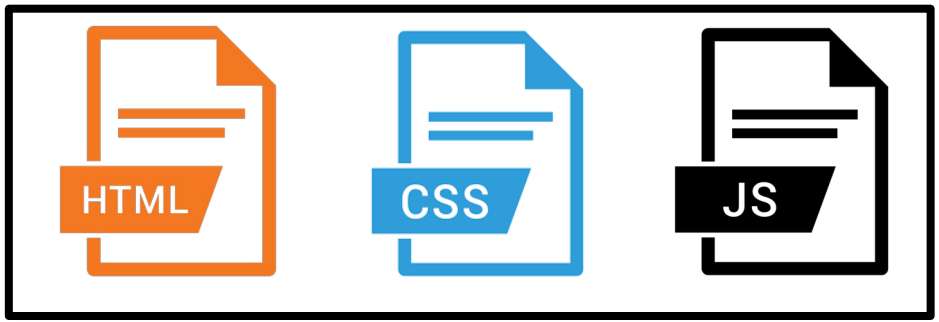
Why do we need VCS?



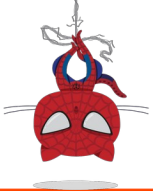
- We may even used a shared folder so other people can see and edit files without sending them over email. Hopefully they relabel the file after they save it.
- Our shared folder/naming system is fine for class projects or one-time papers. But software projects? Not a chance.
- Do you think the Linux Kernel source code sits in a shared folder like “KernelV1.3-Latest-UPDATED!!”, for anyone to edit? That every programmer just works in a different subfolder? No way.

git

What is VCS?



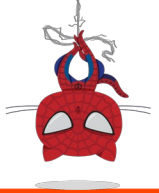
VCS Features



- **Synchronization.** Lets people share files and stay up-to-date with the latest version.
- **Backup and Restore.** Files are saved as they are edited, and you can jump to any moment in time. Need that file as it was on Feb 23, 2007? No problem.
- **Short-term undo.** Monkeying with a file and messed it up? (That's just like you, isn't it?). Throw away your changes and go back to the "last known good" version in the database.
- **Long-term undo.** Sometimes we mess up bad. Suppose you made a change a year ago, and it had a bug. Jump back to the old version, and see what change was made that day.

git

VCS Features

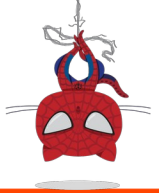


- **Track Changes. As files are updated, you can leave messages explaining why the change happened (stored in the VCS, not the file). This makes it easy to see how file is evolving over time, and why.**
- **Track Ownership. A VCS tags every change with the name of the person who made it. Helpful for blamestorming giving credit.**
- **Sandboxing. or insurance against yourself. Making a big change? You can make temporary changes in an isolated area, test and work out the kinks before "checking in" your changes.**
- **Branching and merging. A larger sandbox.**

  You can branch a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can merge your work back into the common area.

git

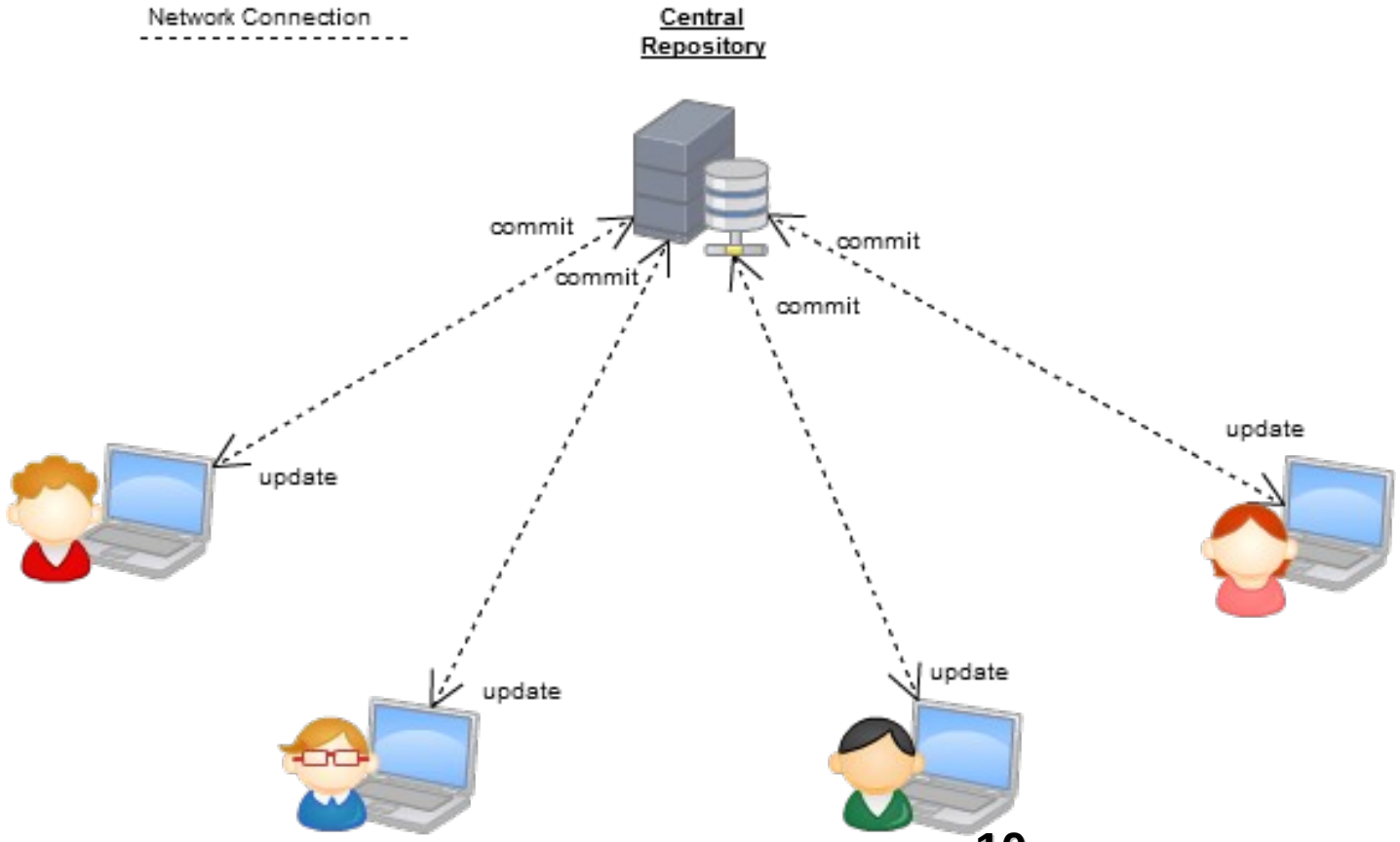
VCS Terminologies

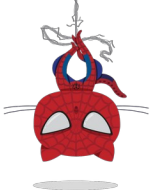


- **Repository (repo):** The database storing the files.
- **Server:** The computer storing the repo.
- **Client:** The computer connecting to the repo.
- **Working Set/Working Copy:** Your local directory of files, where you make changes.
- **Trunk/Main:** The primary location for code in the repo. Think of code as a family tree — the trunk is the main line.

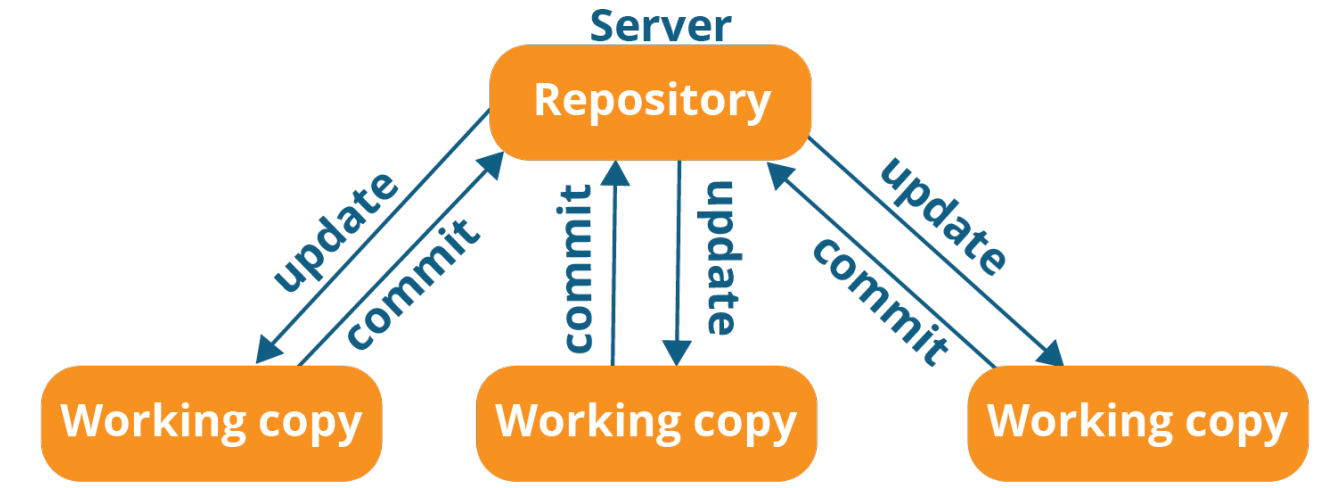
git

Centralized Version Control System



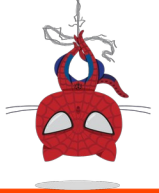


Centralized version control system



git

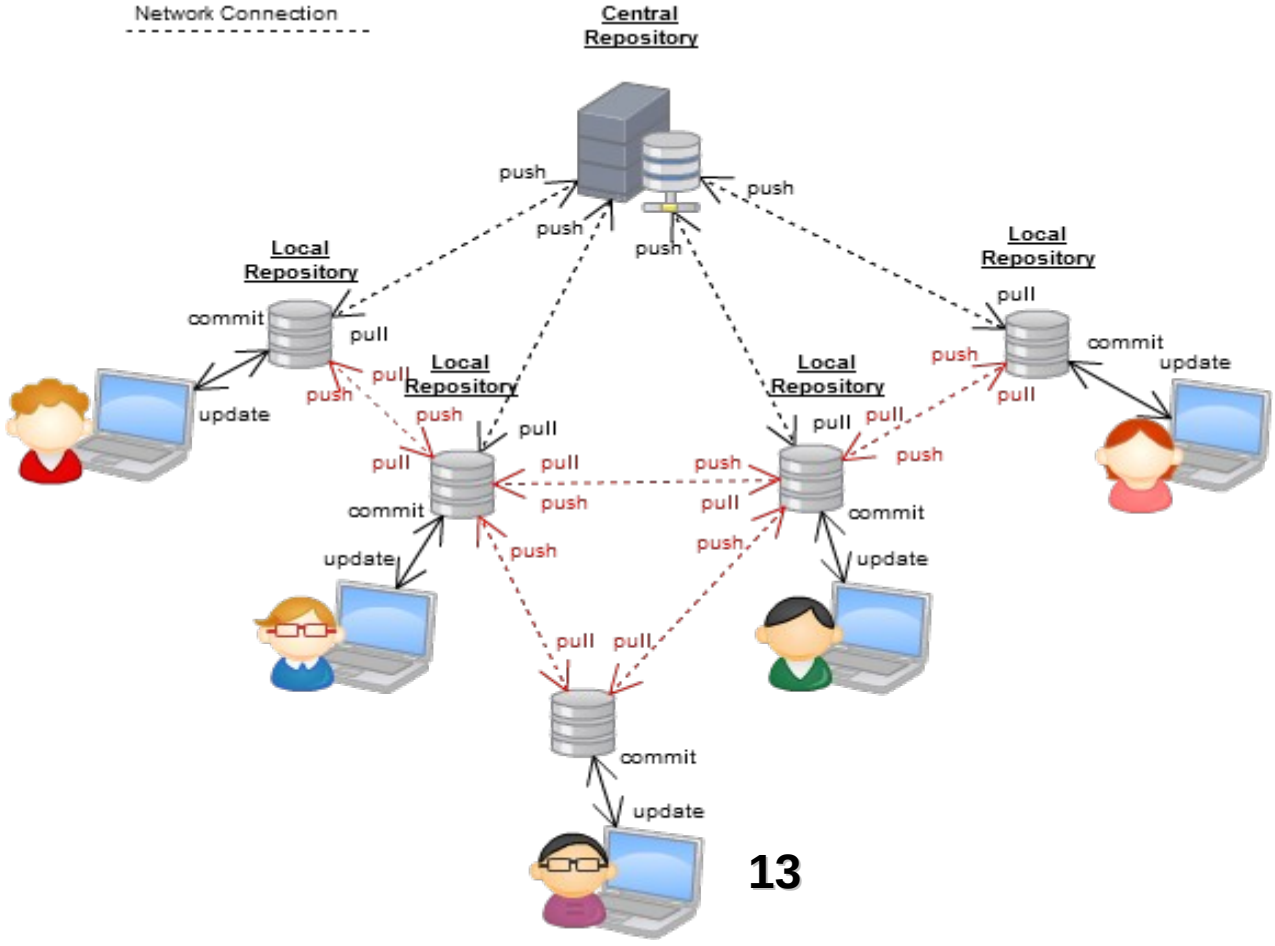
Centralized Version Control System

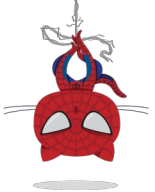


- Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy.
- “Committing” a change simply means recording the change in the central system. Other programmers can then see this change. They can also pull down the change, and the version control tool will automatically update the contents of any files that were changed.

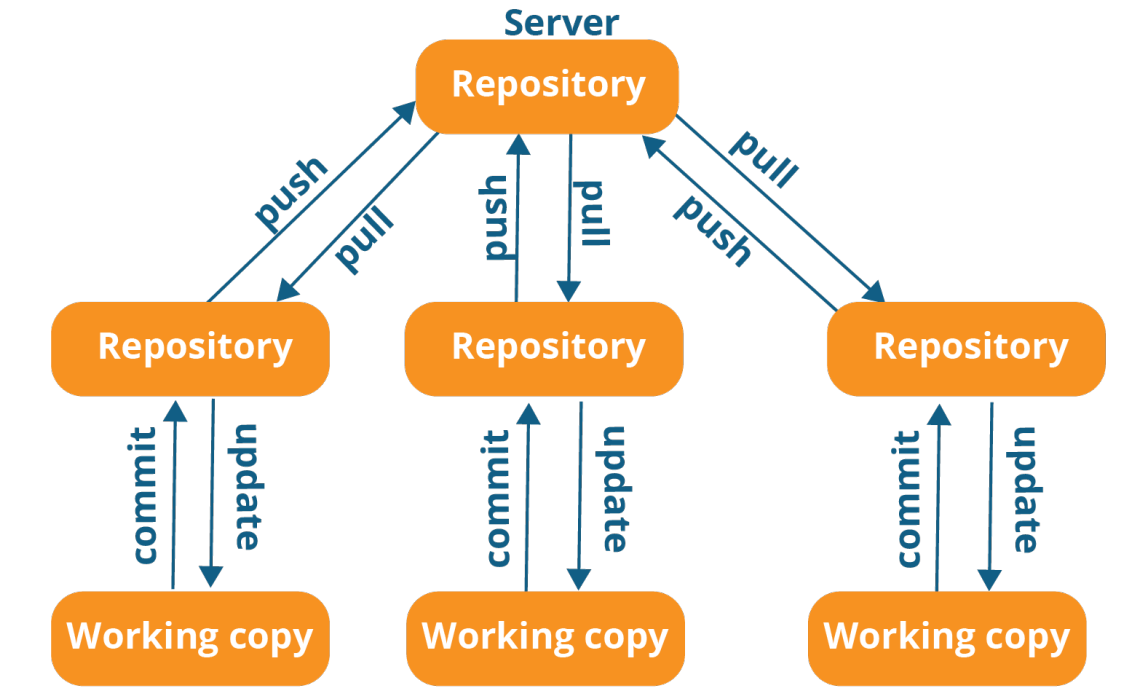
git

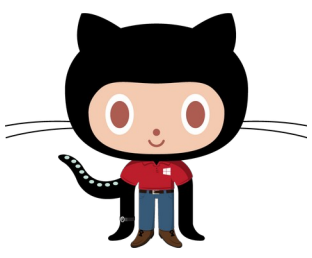
Distributed Version Control System

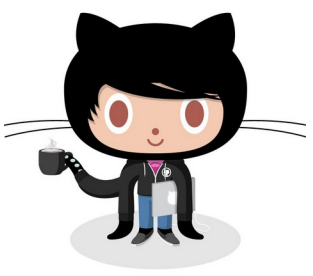




Distributed version control system

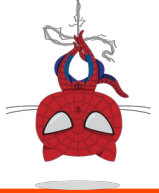






git

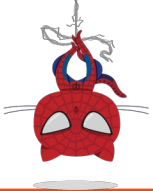
Distributed Version Control System



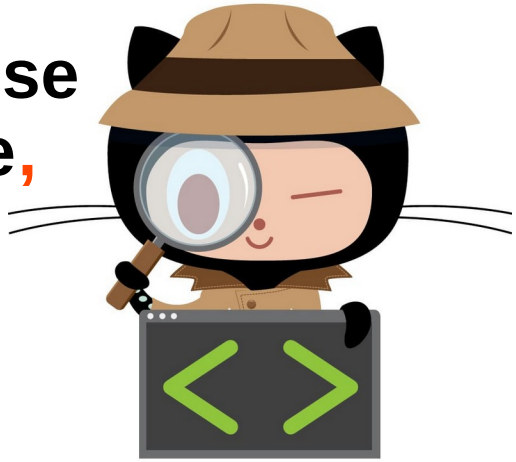
- These systems do not necessarily rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a copy of a repository and has the full history of the project on their own hard drive. This copy (or "clone") has all of the metadata of theoriginal.
- The act of getting new changes from a repository is usually called "pulling," and the act of moving your own changes to a repository is called "pushing". In both cases, you move changesets (changes to files groups as coherent wholes), not single-file diffs.

git

Advantages Over CVCS

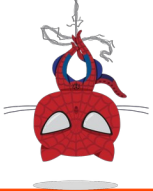


- Performing actions other than pushing and pulling change sets is extremely fast because the tool only needs to access the hard drive, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.

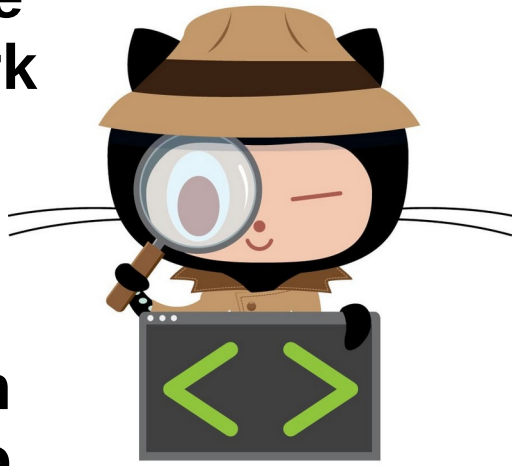


git

Advantages Over CVCS

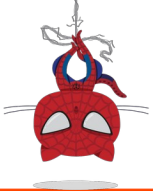


- Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won't be forced to commit several bug fixes as one big changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.



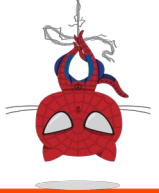
git

Disadvantages Over CVCS

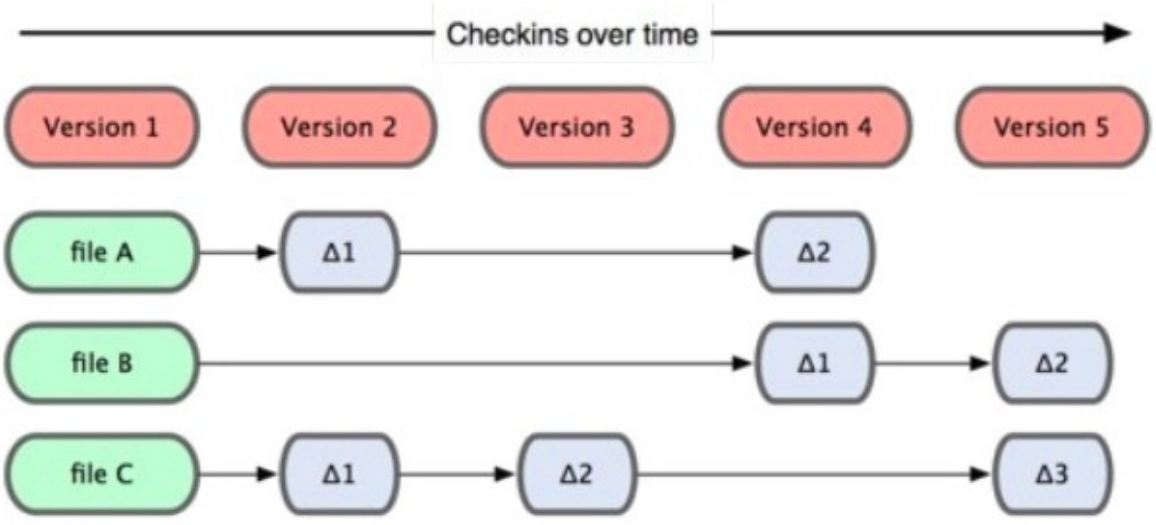


- To be quite honest, there are almost no disadvantages to using a distributed version control system over a centralized one.
- Distributed systems do not prevent you from having a single “central” repository, they just provide more options on top of that.

Snapshots, not Differences

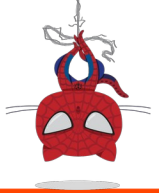


- **Most VCS store information as a list of file-based changes. These systems think of the information they keep as a set of files and the changes made to each file over time.**

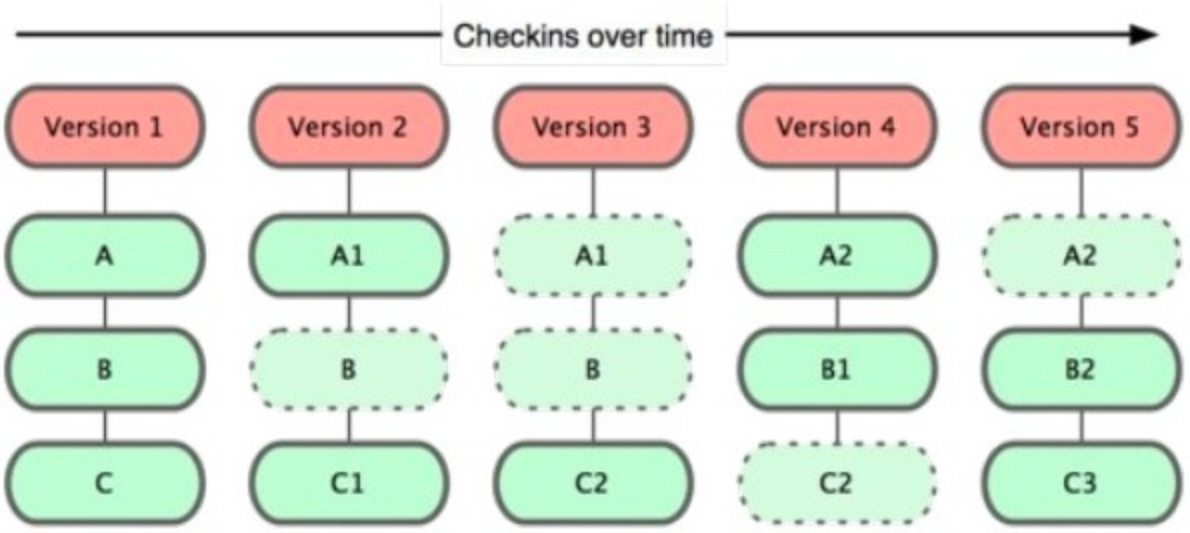


git

Snapshots, not Differences

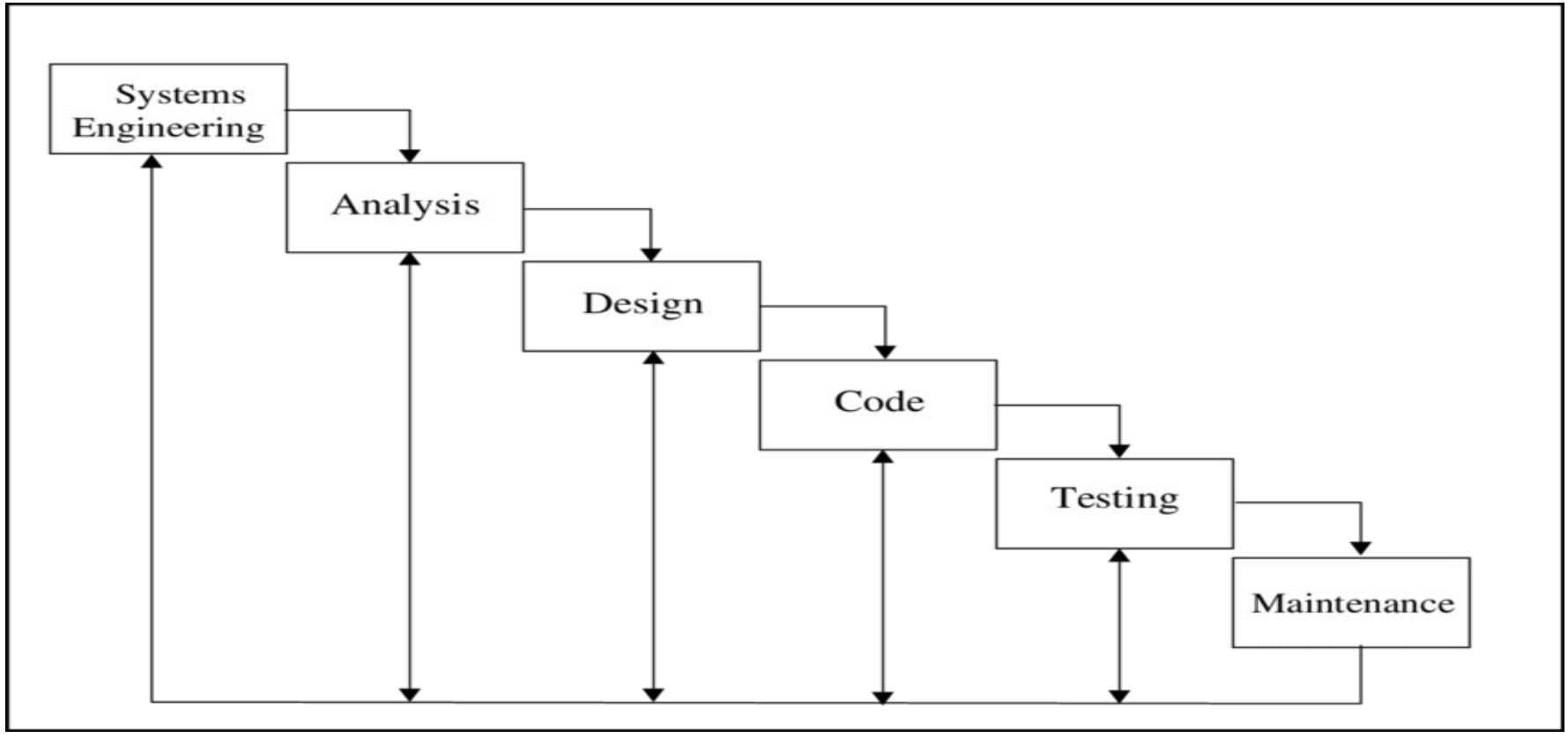


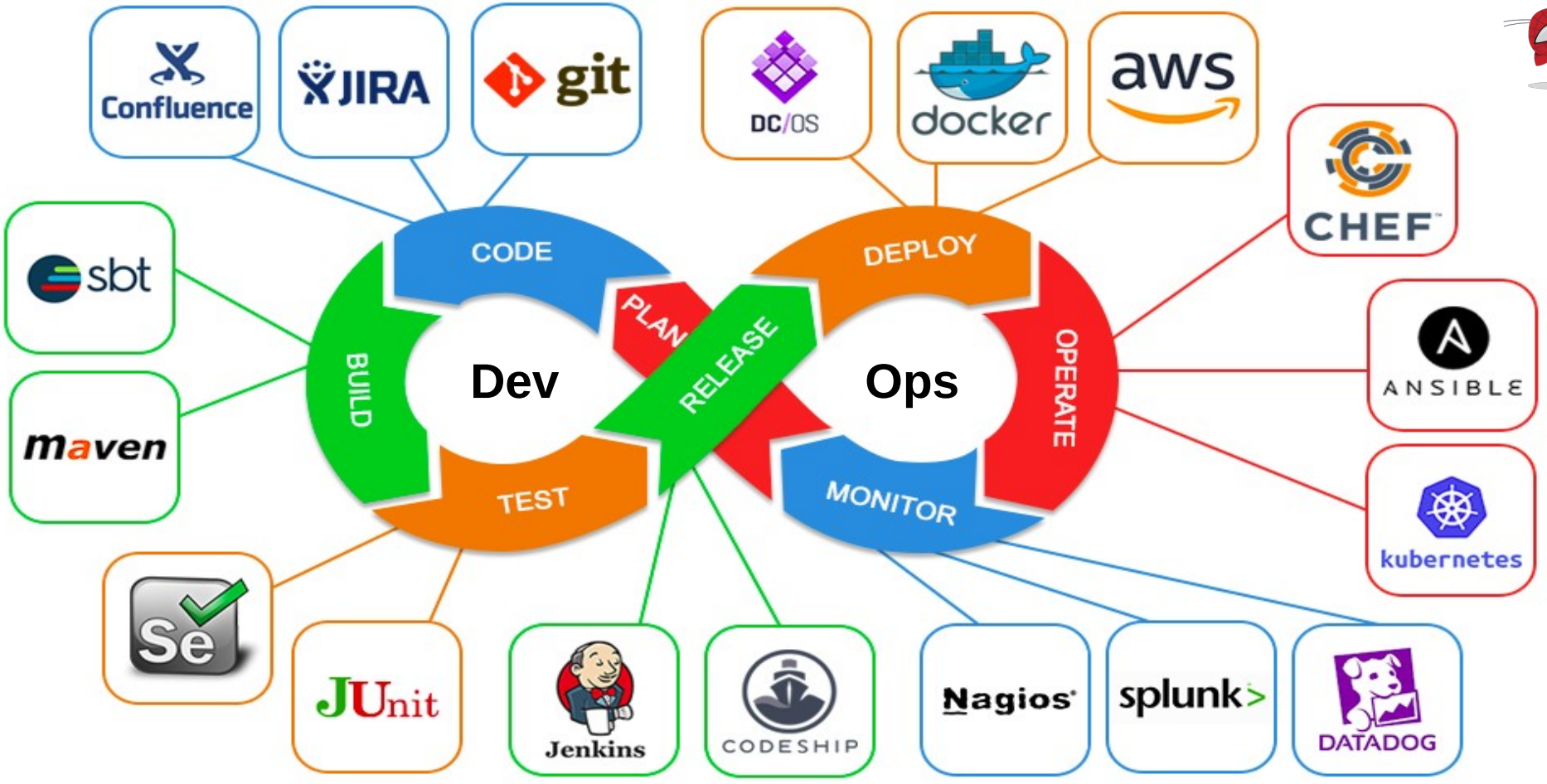
- **Git thinks of its data more like a set of snapshots of a miniature filesystem, Every time you commit in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.**



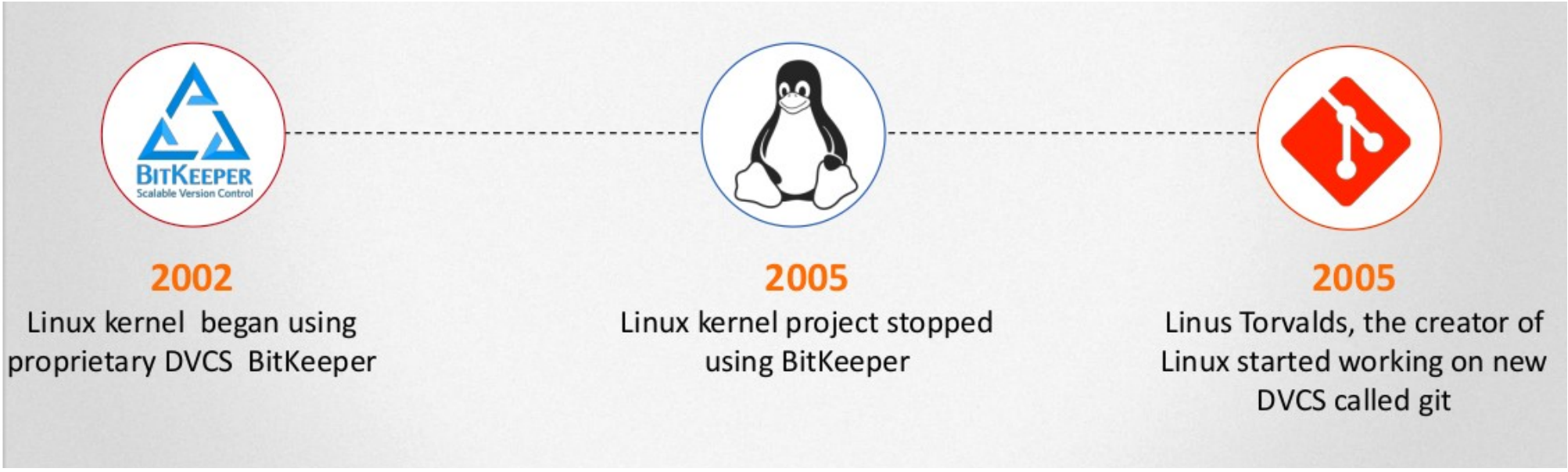
git

Classic Software Life Cycle



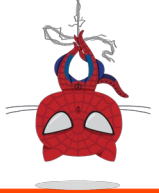

Confluence
JIRA
git
DC/OS
docker
aws
CHEF
sbt
maven
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
Dev
Ops
ANSIBLE
kubernetes
Se
JUnit
Jenkins
CODESHIP
Nagios
splunk>
DATADOG

GIT History



git

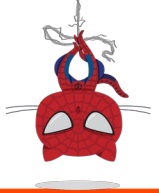
Every Operation Is Local



- Most operations in Git only need local files and resources to operate – generally no information is needed from another computer on your network.
- For example, to browse the history of the project, Git doesn't need to go out to the server to get the history and display it for you – it simply reads it directly from your local database.

git

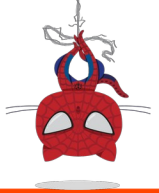
GIT Has Integrity



- Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it.
- The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure.

git

The Three States



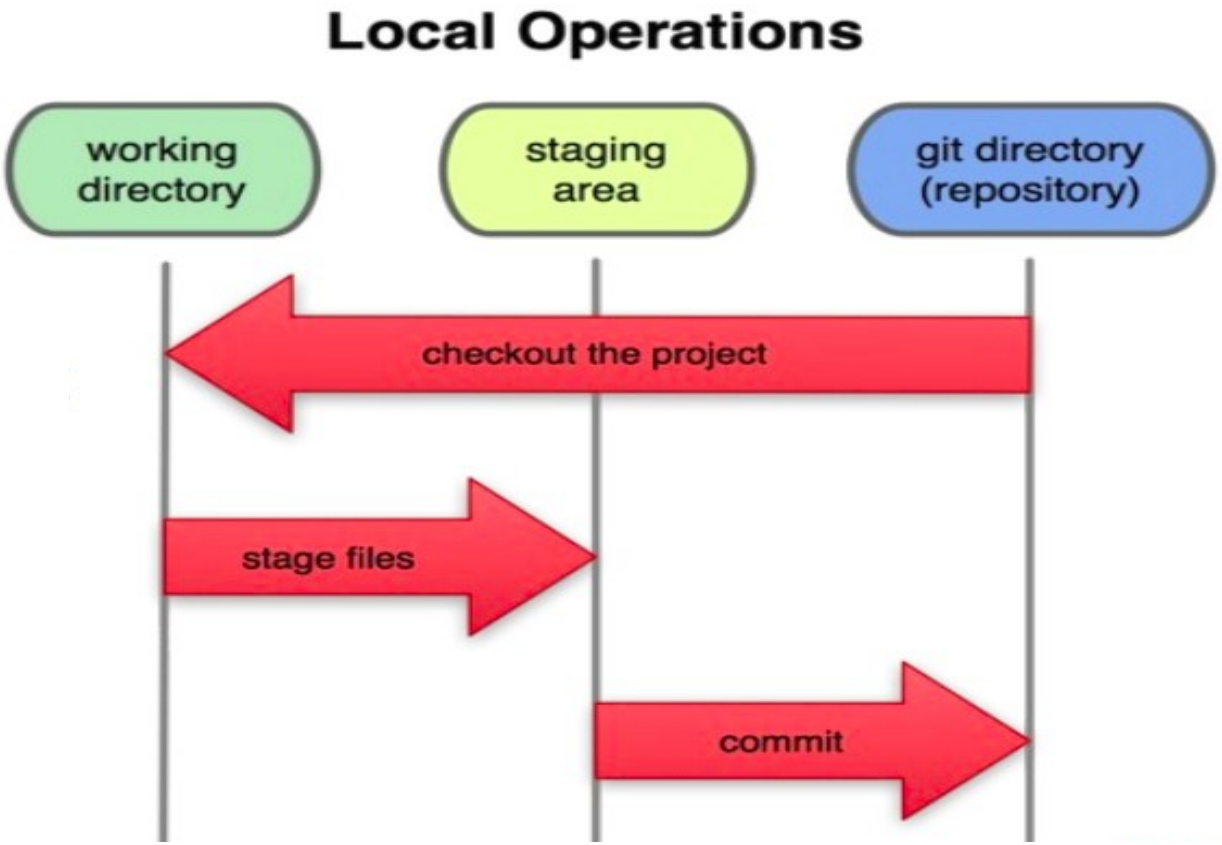
Git has three main states that your files can reside in:

committed, modified, and staged.

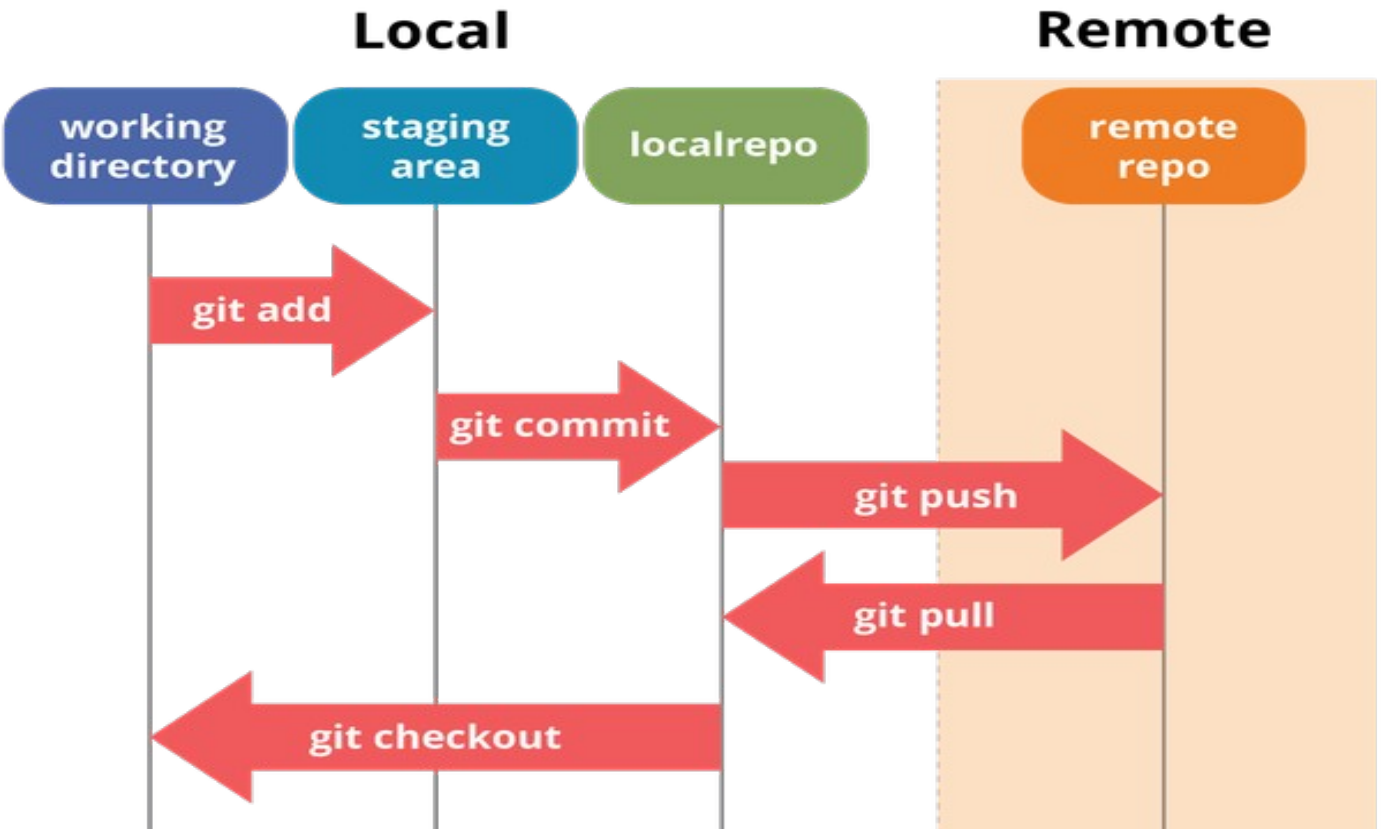
- **Committed means that the data is safely stored in your local database.**
- **Modified means that you have changed the file but have not committed it to your database yet.**
- **Staged means that you have marked a modified file in its current version to go into your next commit snapshot.**

git

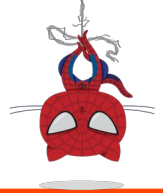
The Three States



The Three States



Installing GIT



- **CentOS**

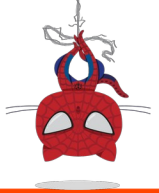
  sudo yum install git-all

- **Ubuntu**

  sudo apt-get install git-all

git

First-Time GIT Setup



- The first thing you should do when you install Git is to set your user name and email address.
- This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating.

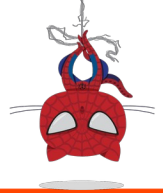
```
git config  --global user.name "Fatma Khaled"

git config  --global user.email  fatma10khaled10@gmail.com
```

- You can also set your default editor and colorize the output:

```
git config  --global core.editor vi

git config  --global color.ui true
```


git

Getting help



- **If you ever need help while using Git, just use any of these commands:**

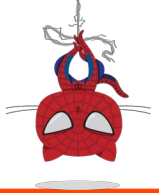
  git help <verb>

  git <verb> --help

  man git-<verb>

git

Getting help

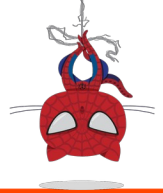


- **git help**

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one
```


git

Starting A Repo



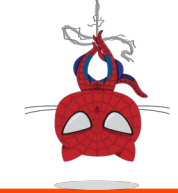
- mkdir gittest
- cd gittest
- git init

  Initialized empty Git repository in /home/fatma/t/gittest/gittest/.git/
- ls -A

  .git

git

GIT Workflow



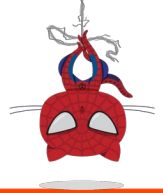
- **Let's assume the following scenario:**

Create A file

touch file1

git

Create File

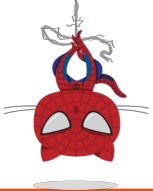


- touch file1
- git status

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1

nothing added to commit but untracked files present (use "git add" to track)
```


git

Add to staging

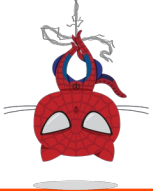


- **git add file1**
- **git status**

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1
```


git

Commit Changes



- **git commit -m "Create File1"**

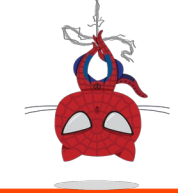
```
[master (root-commit) 14db121] create File1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1
```

- **git status**

```
On branch master
nothing to commit, working tree clean
```


git

GIT Workflow

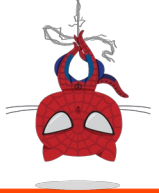


- **Let's assume the following scenario:**

Modifies file1 & **Create file2**

touch file2
vi file1

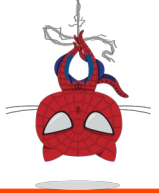
Create File



- touch file2
- vi file1
- git status

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2

no changes added to commit (use "git add" and/or "git commit -a")
```


Add to staging

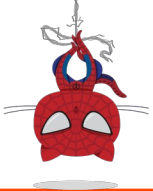


- **git add file1 file2**
- **git status**

```
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1
        new file:   file2
```


git

Commit Changes



- **git commit -m "Create file2 and edit in file1 "**

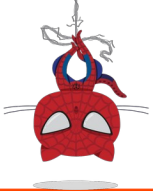
```
[master 6da5297] Create file2 and edit file1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f3
```

- **git status**

```
On branch master
nothing to commit, working tree clean
```


git

Git Log



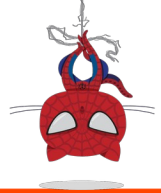
- **git log**

```
commit 6da529712906478c317a86ef139326df2bd18bde (HEAD -> master)
Author: FatmaKhaled <fatma10khaled10@gmail.com>
Date:   Wed Oct 28 20:17:16 2020 +0200

    Create file2 and edit file1
```

```
commit 14db12122a3e72783a2f4a7293e51e63e29ce35e
Author: FatmaKhaled <fatma10khaled10@gmail.com>
Date:   Wed Oct 28 19:24:55 2020 +0200

    create File1
```


git

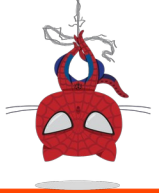
Different Ways to Add



- git add <list of files>
- git add --all
- git add dir/
- git add *.rb

git

Git Diff



- **Show unstaged differences since last commit**

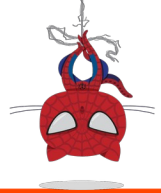
  git diff

- **Show staged differences since last commit**

  git diff --staged

git

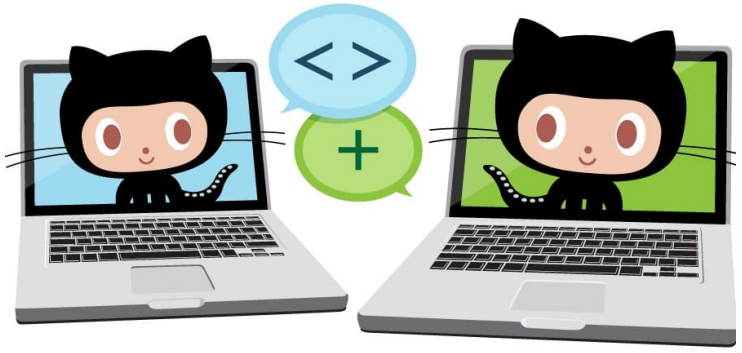
Discard changes



- git status –s

  M file1

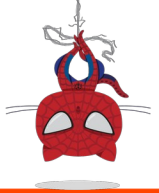
- git checkout –- file1
- git status



```
On branch master
nothing to commit, working tree clean
```


git

Unstaging Files



- **To unstage file**

  git reset HEAD file1

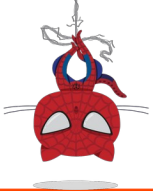
- **Short Status**

  git status –s

  M file1

git

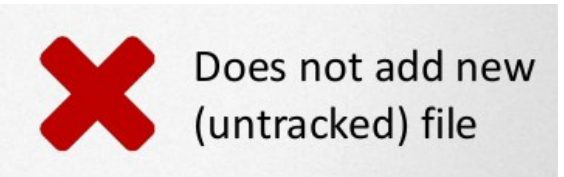
Skip Staging



- **git commit –a –m “modify file1.”**

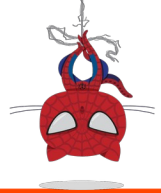
  [master d00fefc] Modify file1.

  1 files changed, 1 insertions(+),1 deletions(-)



git

Undoing A Commit

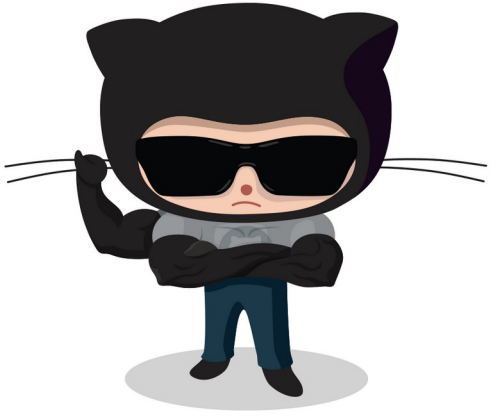


- Move to commit before ‘HEAD’ (last commit)

  git reset --soft HEAD^
- Undo last 2 commits

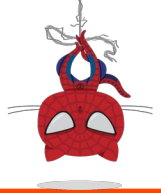
  git reset --soft HEAD^^
- Undo the last commit and all changes

  git reset --hard HEAD^



git

Adding TO A Commit



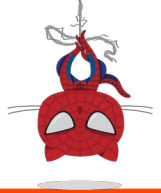
- **Maybe you forgot to add a file**

  git add file3

  git commit --amend -m "Modify file2 & add file3."

git

Adding/Removing A Remote



- To rename The master branch to New branch

  git branch -M main

- To add remote repository

  git remote add origin https://github.com/FatmaKhaled/opensource

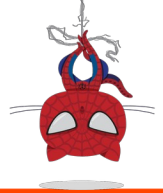
- To list the remote repositories

  git remote –v

```
origin  https://github.com/FatmaKhaled/opensource (fetch)
origin  https://github.com/FatmaKhaled/opensource (push)
```


git

Adding/Removing A Remote



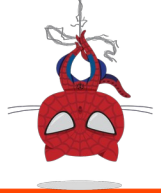
- **To delete or to rename a remote repository:**

  git remote rm origin

  git remote rename origin neworigin

git

Pushing / Pulling From Remote



- To add your files to the remote repo use this command (Note: you need to create an account on github.com)

  git push origin main

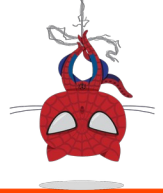
- To get the changes made by other

  git pull origin main

- this command will:
  - Fetch (or Sync) our local repository with the remote one.
  - Merge the origin/main with main.

git

Dealing With Conflicts



- **We have to edit the file and fix the conflict:**

  <<<<<<< HEAD

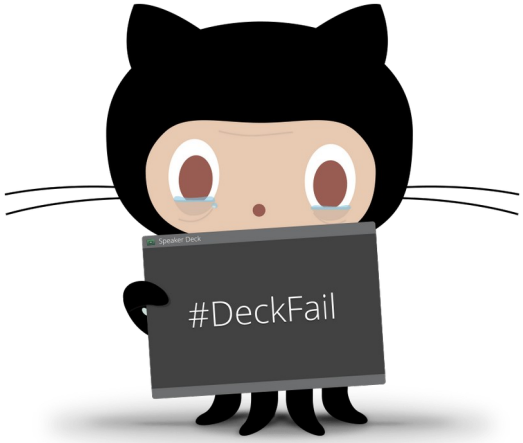
  This my line.

  =======

  No, it's mine!

  >>>>>>>

  4e76d3542a7eee02ec516a47600002a90a4e4b48

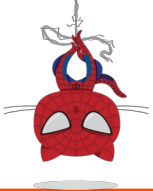


- **Then commit to save the merge, then the editor will pop up to enter your commit message**

  git commit -a

git

Cloning A Repository



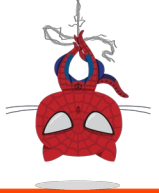
- **To clone a repo**

  git clone https://github.com/FatmaKhaled/opensource

- **This will:**
  - **Download the entire repository into a new opensource directory.**
  - **Add the ‘origin’ remote, pointing it to the clone URL.**

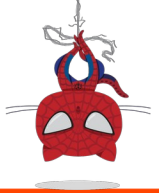
git

VCS Terminologies



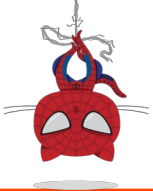
- **Add:** Put a file into the repo for the first time, i.e. begin tracking it with Version Control.
- **Revision:** What version a file is on (v1, v2, v3, etc.).
- **Head:** The latest revision in the repo.
- **Check out:** Download a file from the repo.
- **Check in (Commit/Push):** Upload a file to the repository (if it has changed). The file gets a new revision number, and people can "check out" the latest one.
- **Checkin Message:** A short message describing what was changed.

VCS Terminologies



- **Changelog/History:** A list of changes made to a file since it was created.
- **Update/Sync/Pull:** Synchronize your files with the latest from the repository. This lets you grab the latest revisions of all files.
- **Revert:** Throw away your local changes and reload the latest version from the repository.
- **Branch:** Create a separate copy of a file/folder for private use (bug fixing, testing.
- **Diff/Change/Delta:** Finding the differences between two files. Useful for seeing what changed between revisions.

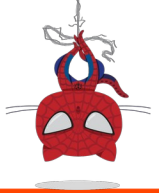
VCS Terminologies



- **Merge (or patch):** Apply the changes from one file to another, to bring it up-to-date. For example, you can merge features from one branch into another.
- **Conflict:** When pending changes to a file contradict each other (both changes cannot be applied).
- **Resolve:** Fixing the changes that contradict each other and checking in the correct version.
- **Locking:** Taking control of a file so nobody else can edit it until you unlock it. Some version control systems use this to avoid conflicts.
- **Breaking the lock:** Forcibly unlocking a file so you can edit it. It may be needed if someone locks a file and goes on vacation.

git

Lab 1



- **Install Git.**
- **Create an account on Github and bitbucket.**
- **Accept my invitation in opensource repository from gmail. Then clone it to your machine, and upload file with your full name.**
- **Create a local repository on your machine, and create a remote repository in github (and bitbucket). Then Upload a file with your full name from your local repository to remote repository. And send me an invitation.**
- **Report: What is the Pull Request.**