Version Control with Git

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University


beautiful!
what do you call it?
mona_lisa_finalrealupdateFINALL6

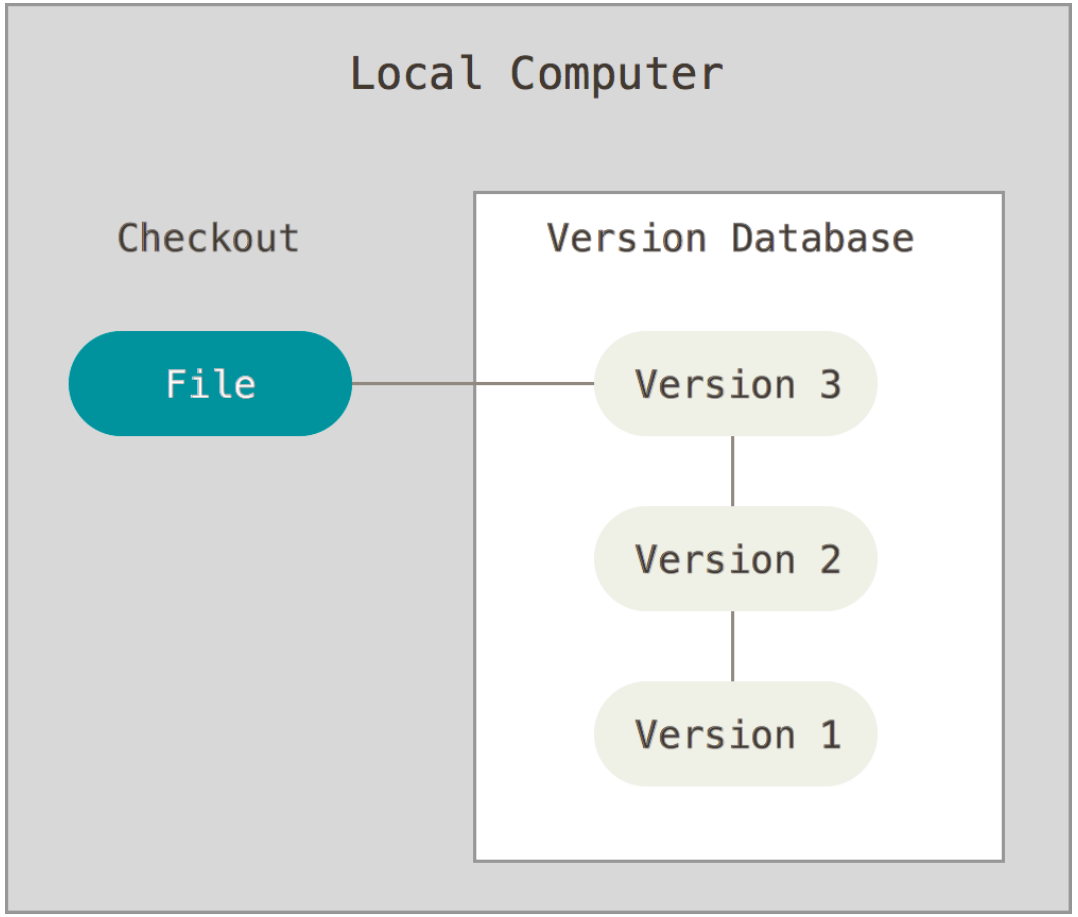
What is Version Control & Why should you care?

- Version Control System track changes to your files and folders over time so that you can recall specific versions later.
- With a Version Control System you can compare various revisions of a file or revert back to an older version if there is a problem with latest version of a file.
- Version Control System also lets you track WHO made the changes for a particular version which is useful if you are working in teams.

Local Version Control

Example:

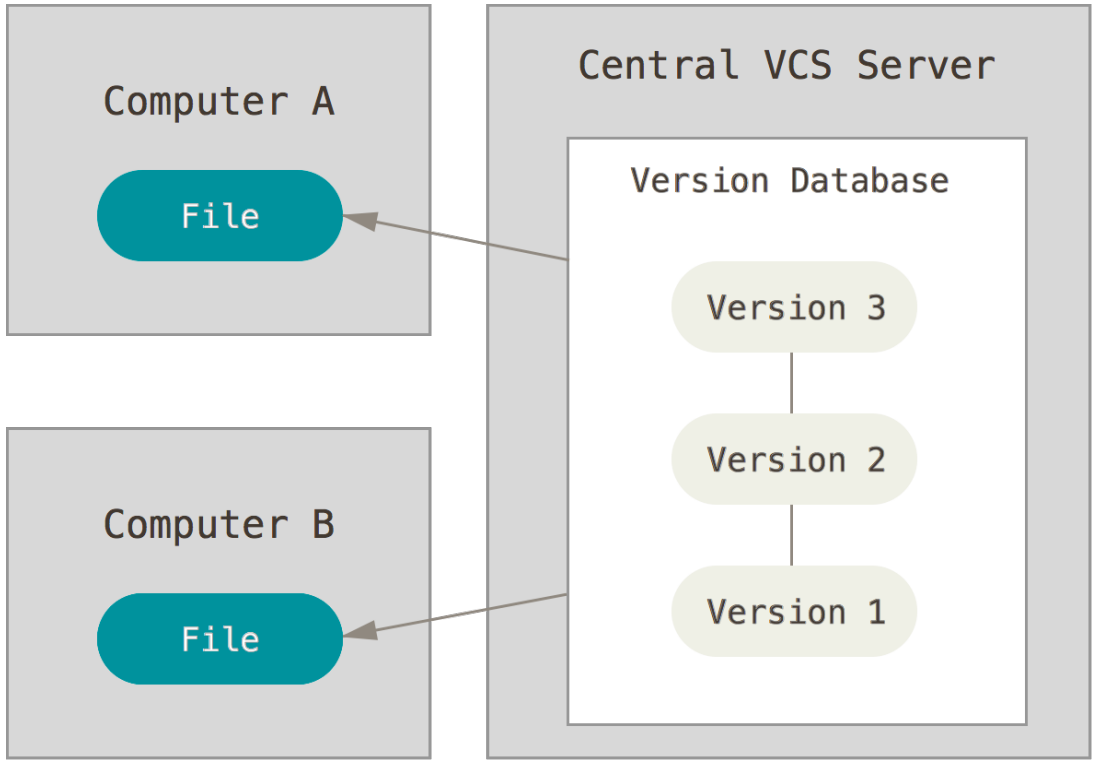
- Revision Control System (RCS)
- Source Code Control System (SCCS)



Centralized Version Control

Example:

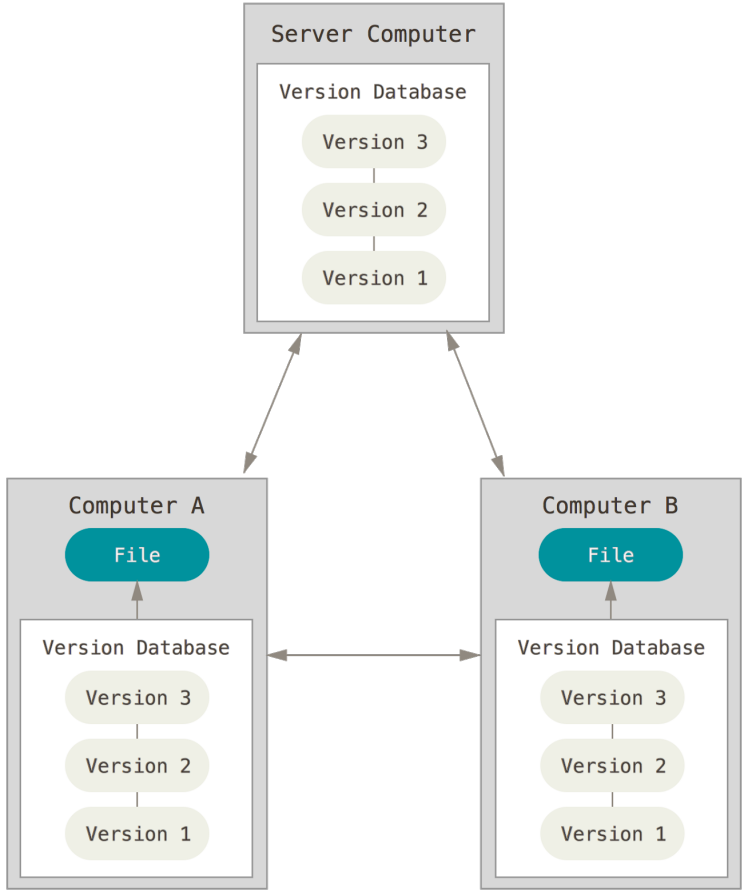
- CVS
- SVN
- Perforce



Distributed Version Control Systems

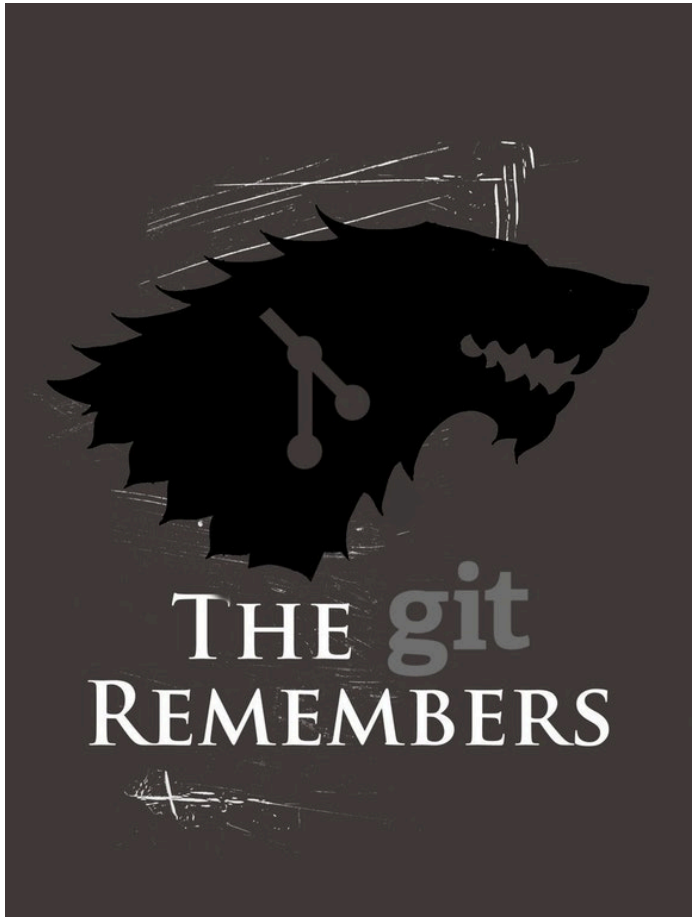
Example

- Git
- Mercurial
- Bazaar
- Darcs

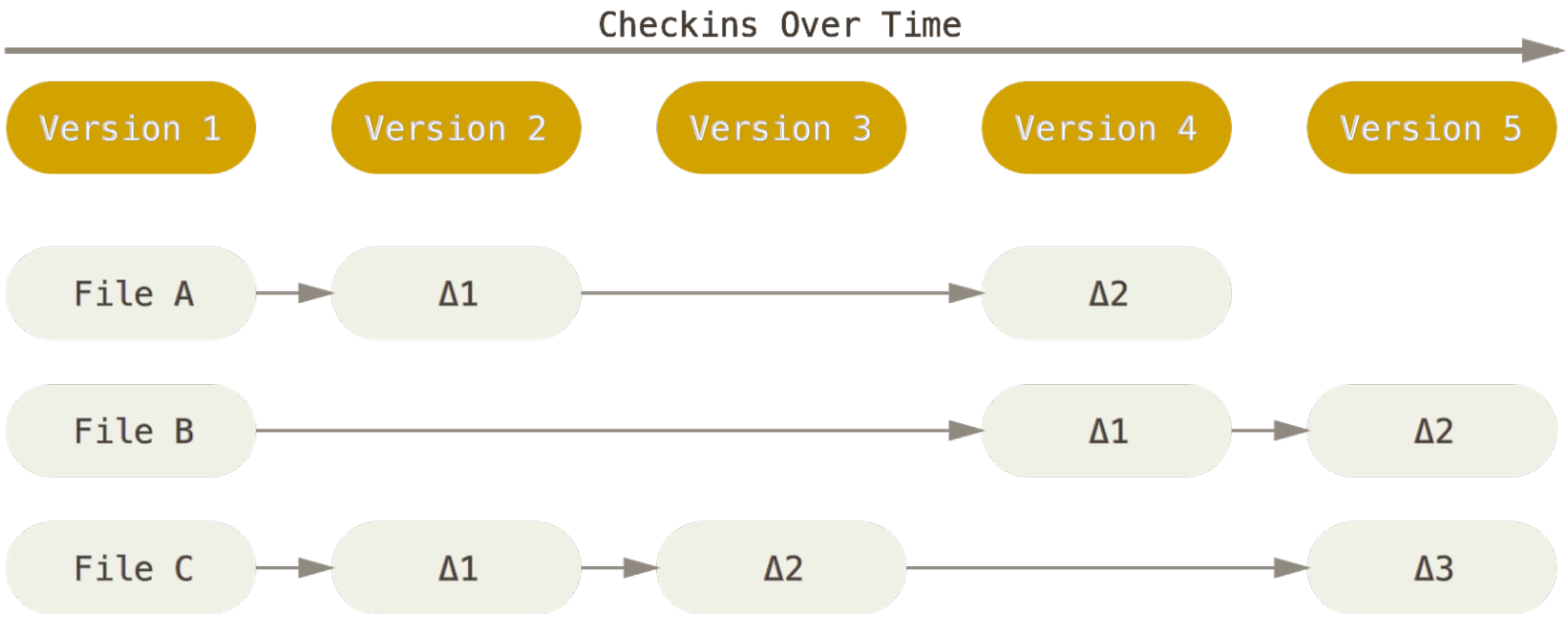


A Short History of Git

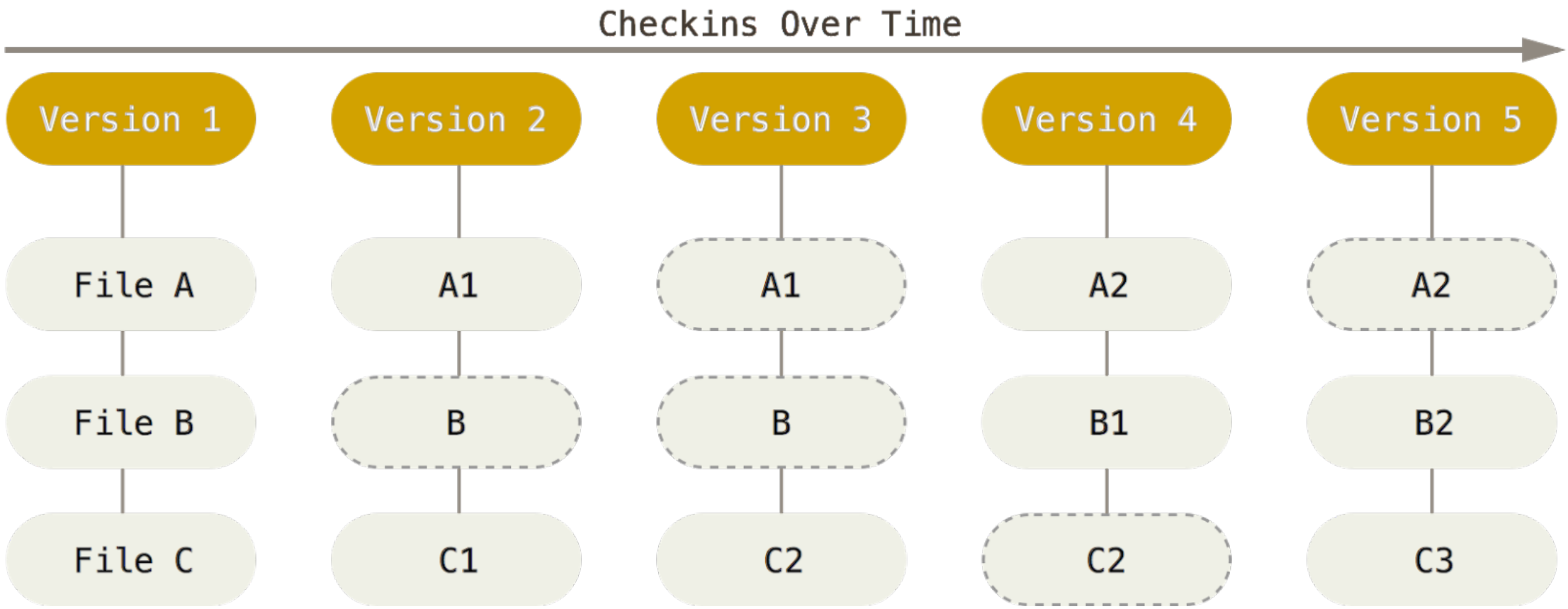
- Linux kernel project began using a proprietary DVCS called BitKeeper in 2002 but in 2005 the relationship between community & commercial company broke down and BitKeeper was no longer free for the community.
- Linus Torvalds created Git in 2005 to manage Linux Kernel. Since then Git has become the goto Distributed Version Control System for developers.
- In this course we will use Git and Github for hosting assignment and term project.



Tracking Changes w/Differences (CVS, SVN)



Tracking Changes w/Snapshots (Git)



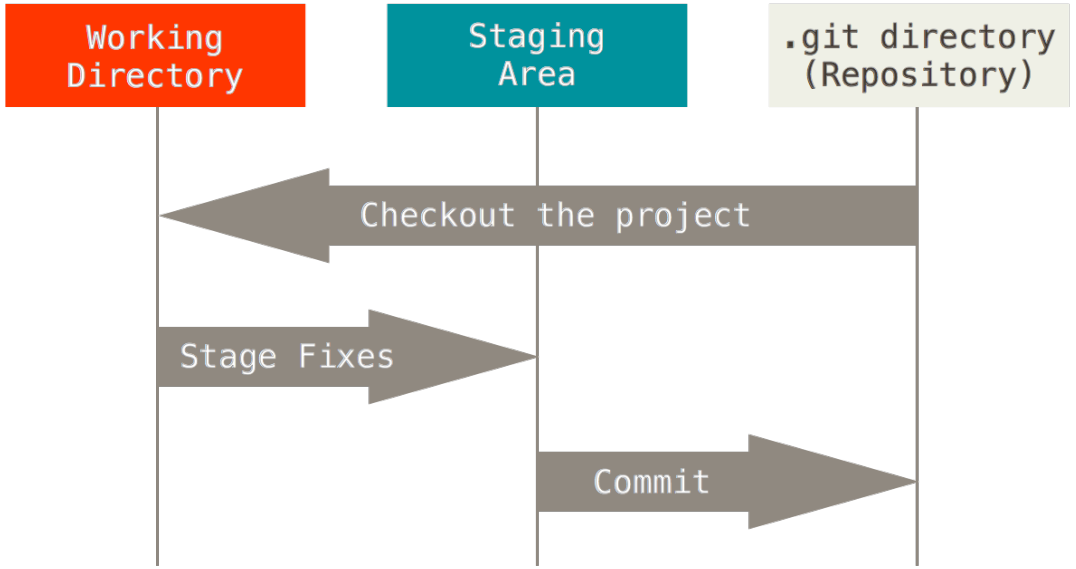
Git Has Integrity

- Everything in Git is check-summed before it is stored and is then referred to by that checksum.
- Once a file has been committed, it is impossible to change the contents of file or directory with Git finding out.
- Git uses SHA-1 hash which 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 in Git. Example commit hash *fb1d8e0e2c50f374cfc244564decfc3f0a336cb4*
- Git stores everything in its database not by file name but by the hash value of its contents which means you will see these hash values all over the place.

Local Git Workflow

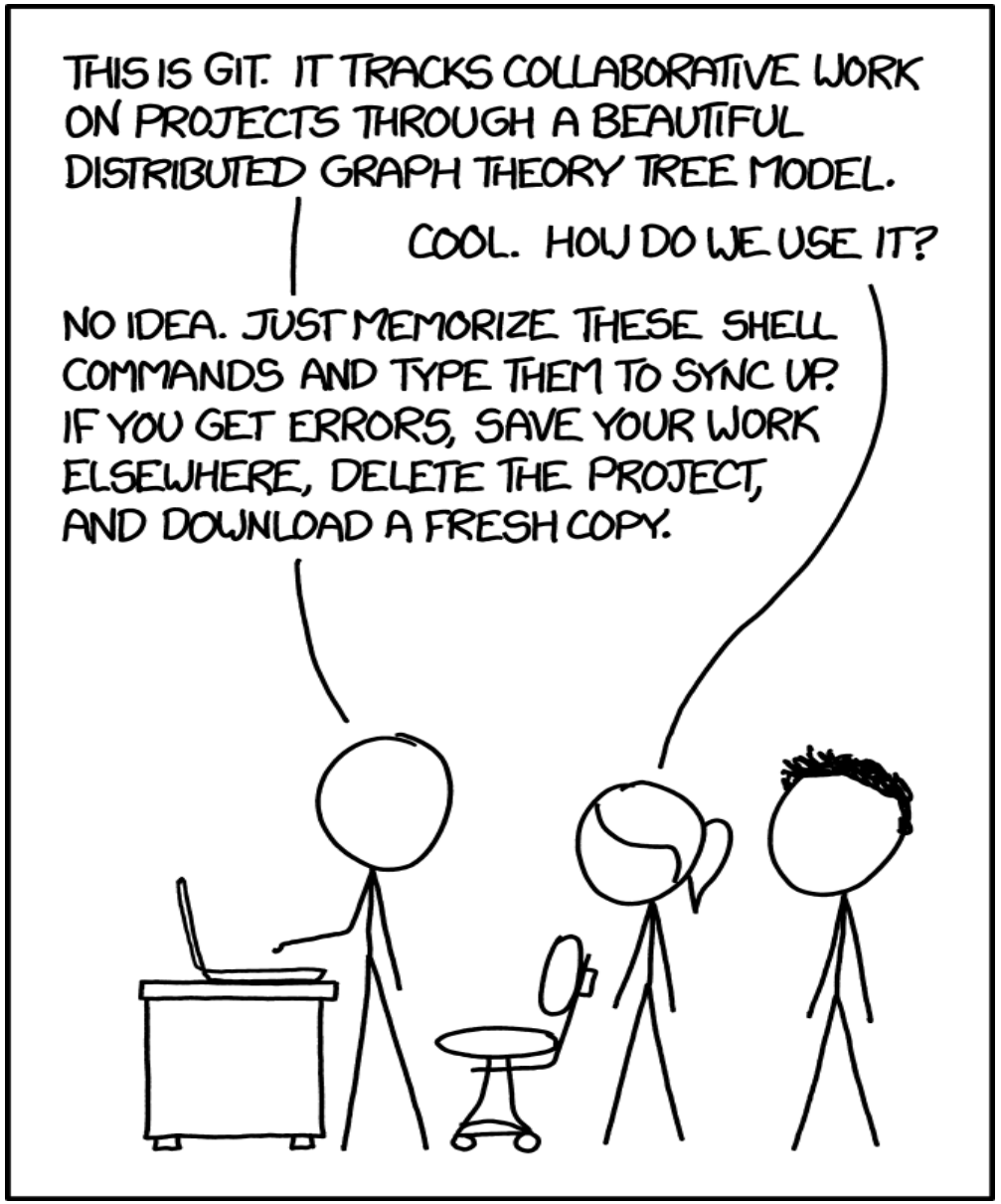
The basic Git workflow is something like this:

- You modify files in your working tree.
- You stage the files, adding snapshots of them to your staging area.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.



Installing Git

You can download the Git binaries from https://git-scm.com/downloads



Source: https://xkcd.com/1597/

First-Time Git Setup (your identity & editor)

Set your name and address using following commands

- ***$ git config --global user.name "John Doe"***
- ***$ git config --global user.email johndoe@example.com***

Set your default editor on Linux using following command.

- ***$ git config --global core.editor vim***
- The default text editor that will be used when Git needs you to type in a message.

You can modify the settings later in ***~/.gitconfig***

GitHub SSH key Setup

Follow steps documented in the articles below:

1. https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/
2. https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

Initializing a Repository in Existing Directory

$ cd /home/user/your_repo

$ git init

This creates a new subdirectory named **.git** that contains all of your necessary repository files – a Git repository skeleton. At this point, nothing in your project is tracked yet.

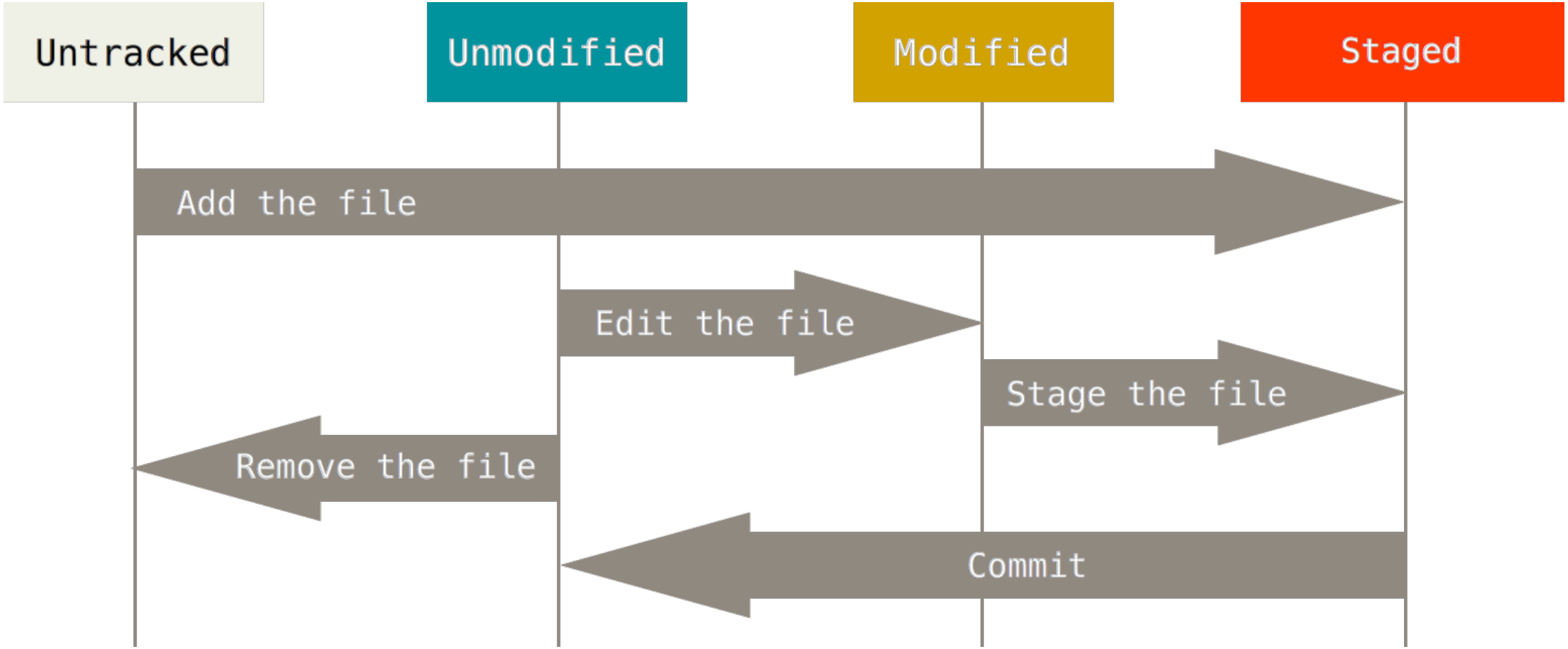
Cloning an Existing Repository

- If you want to get a copy of existing Git repository you need to use **git clone**

$ git clone git@github.com:torvalds/linux.git

- Note that you can clone git repository using either **git** (SSH) or **https** transfer protocol. I recommend you use **git** protocol wherever possible.

Lifecycle of the Status of your files



Git Status

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git.

```
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
tejas@csye6225:~/vimrc$ touch test.txt
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        test.txt

nothing added to commit but untracked files present (use "git add" to track)
tejas@csye6225:~/vimrc$ vi vimrc
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   vimrc

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

        test.txt

no changes added to commit (use "git add" and/or "git commit -a")
tejas@csye6225:~/vimrc$
```

No changes

Added new file

Modified existing file

Tracking New File with Git

A new file must be added to Git repo using the command **git add <FILENAME>**. You can track all new files using **-A**

Example:

$ git add –A :/

Stage Modified Files

- Existing files must be staged using the same **git add** command.

Committing Your Changes (Locally)

Once you have staged all your new and modified files, it is time to commit them using the **git commit** command.

$ git commit -m “commit message goes here”

https://xkcd.com/1296/

Removing Files

- To remove a file from git you must remove it from your staging area and then commit it.
- Git command to remove a file from staging area is **git rm FILENAME**
- Git commit will then remove it from your git repository.

Moving Files

- You can move git files while preserving history using command **git mv from_file to_file**
- Once you have moved the file must stage and commit your changes.

Working with Remotes

- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- To add a new remote Git repository use following command
- **git remote add <url>**
- Note: A git repository can have more than one remote.

Showing Your Remotes

- **git remote** command will show you which remote server you have configured.

“git pull a day keeps the conflicts away”

Fetching and Pulling from Your Remotes

- To get data (changes) from remote project, you can use **git fetch** or **git pull** command.
- **git fetch** command only downloads the data to your local repository – it doesn't automatically merge it with any of your work or modify what you're currently working on. You have to merge it manually into your work when you're ready.
- **git pull** command to automatically fetch and then merge that remote branch into your current branch.

Pushing to Remotes

git push [remote-name] [branch-name] command is used to push committed changes from your local git repository to the one the server so others can pull it.

Example:

$ git push origin master

Git Branches

Nearly every VCS has some form of branching support. Branching means you diverge from the main line of development and continue to do work without messing with that main line. In many VCS tools, this is a somewhat expensive process, often requiring you to create a new copy of your source code directory, which can take a long time for large projects.

Create & Checkout Git Branch

To create a branch and switch to it at the same time, you can run the git checkout command with the -b switch

$ git checkout -b BRANCH_NAME

This is shorthand for:

$ git branch BRANCH_NAME

$ git checkout BRANCH_NAME

Github Pull Request

https://help.github.com/articles/about-pull-requests/

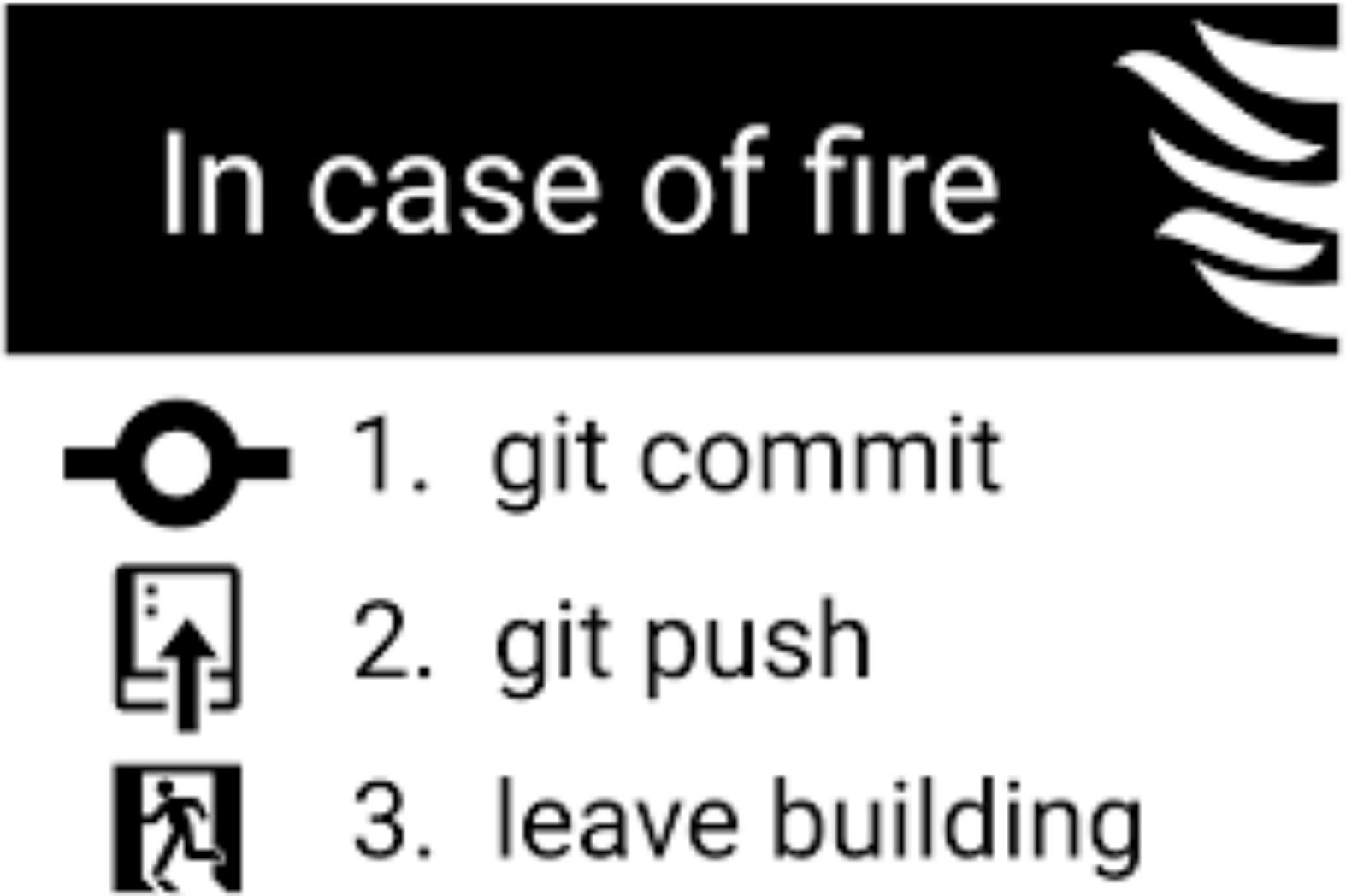
Merge Conflicts

I Am Devloper
@iamdevloper

ever see a git merge conflict so bad you're convinced Netflix are gonna make a true crime mini-series out of it?

9/7/18, 6:14 AM


GIT MERGE


In case of fire
1. git commit
2. git push
3. leave building

Forking Workflow

https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

Additional Resources

https://summer2019.csye6225.cloud/