Introduction to Git

Cengiz Kandemir

November 24, 2021

Who am I?

Currently: Software Engineer @ Sioux, working w/ Philips IGT

Before:

- Software Engineer @ AirTies
- Research Assistant @ EMU (co-supervised by Cem Kalyoncu)

Outline

What is Git?

Git is software for **tracking changes** *in any set of files, usually used for* **coordinating work** *among programmers collaboratively developing source code during software development.*

Wikipedia

Why do developers need a version control systems?

- Synchronize changes made by two or more developers

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility
  - Traceability: Bugs can be narrowed down based on the locality of changes

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility
  - Traceability: Bugs can be narrowed down based on the locality of changes
  - Visibility: Changes are visible to other stakeholders
    - Improved collaboration

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility
  - Traceability: Bugs can be narrowed down based on the locality of changes
  - Visibility: Changes are visible to other stakeholders
    - Improved collaboration
- Improved software quality

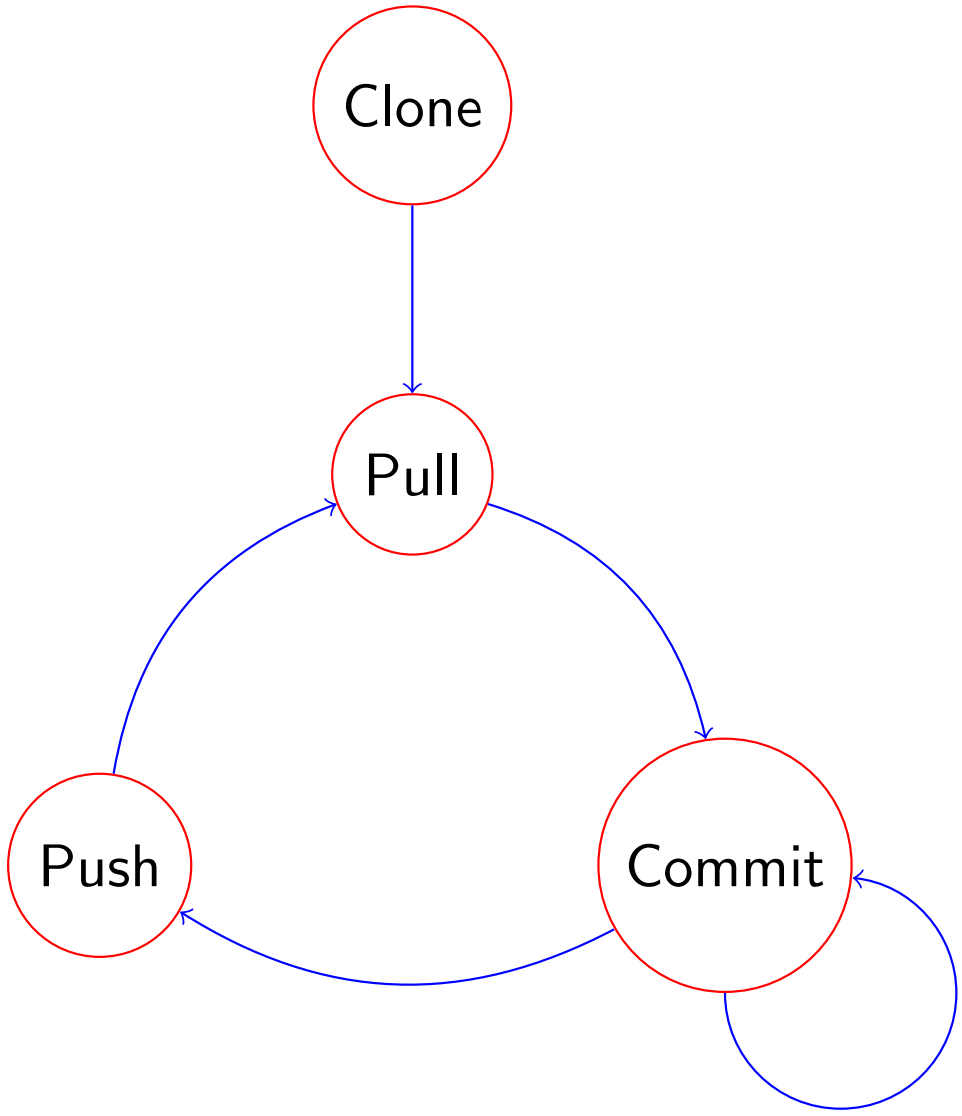
Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility
  - Traceability: Bugs can be narrowed down based on the locality of changes
  - Visibility: Changes are visible to other stakeholders
    - Improved collaboration
- Improved software quality
  - Testing/Static analysis

Why do developers need a version control systems?

- Synchronize changes made by two or more developers
- Undo/redo changes conveniently
- Bookmark (aka *tag*) specific points in a repository's history
  - Releases/important milestones
  - Critical/risky changes
- Improved traceability and visibility
  - Traceability: Bugs can be narrowed down based on the locality of changes
  - Visibility: Changes are visible to other stakeholders
    - Improved collaboration
- Improved software quality
  - Testing/Static analysis
  - CI/CD pipelines

Git loop



git-clone

- Clones a git repository (*.git* folder)

git-clone

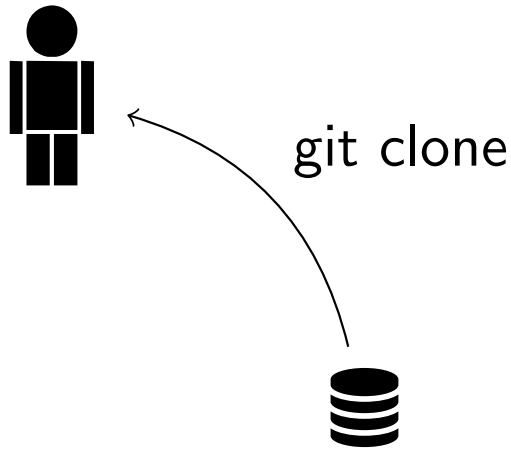
- Clones a git repository (*.git* folder)
- A repository can be located **locally** or **remotely**

git-clone

- Clones a git repository (*.git* folder)
- A repository can be located **locally** or **remotely**

  git clone <path-to-repo>

git-clone cont.



git-add & git-commit

- A commit is composed of two steps: staging and committing

git-add & git-commit

- A commit is composed of two steps: staging and committing
- Staging is useful for compartmentalizing changes

git-add & git-commit

- A commit is composed of two steps: staging and committing
- Staging is useful for compartmentalizing changes

  git add <*list-of-files-to-be-staged*>

  git stage does the same

  git commit

git-add & git-commit

- A commit is composed of two steps: staging and committing
- Staging is useful for compartmentalizing changes

  git add <list-of-files-to-be-staged>

  git stage does the same

  git commit
- The state of repository after a commit is recorded and can always be returned to

git-add & git-commit

- A commit is composed of two steps: staging and committing
- Staging is useful for compartmentalizing changes

  git add <list-of-files-to-be-staged>

  git stage does the same

  git commit
- The state of repository after a commit is recorded and can always be returned to
- Committing a set of changes creates a “bookmark“, identified by a **commit hash**

git-add & git-commit

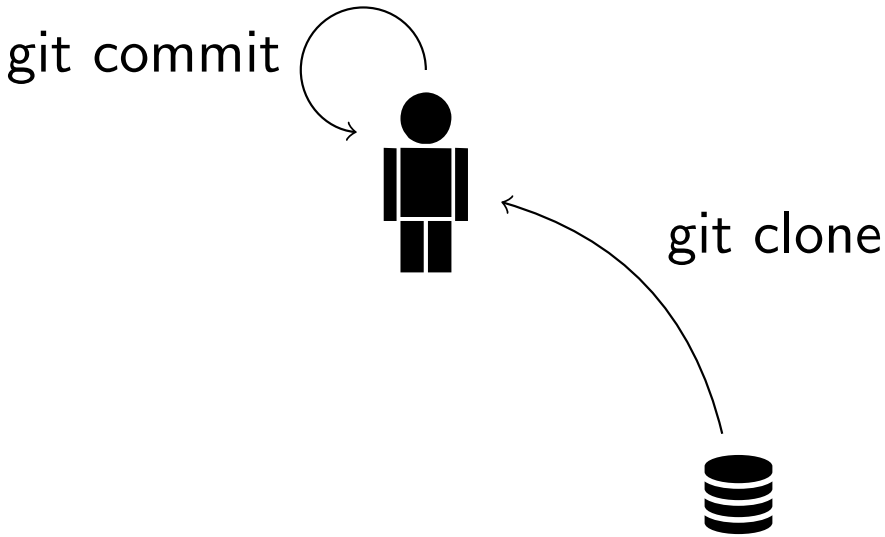
- A commit is composed of two steps: staging and committing
- Staging is useful for compartmentalizing changes

  git add <list-of-files-to-be-staged>

  git stage does the same

  git commit
- The state of repository after a commit is recorded and can always be returned to
- Committing a set of changes creates a "bookmark", identified by a **commit hash**
- Many commands in Git require a commit hash as an input

git-add & git-commit cont.



git-push

- Updates remote/server with the local changes

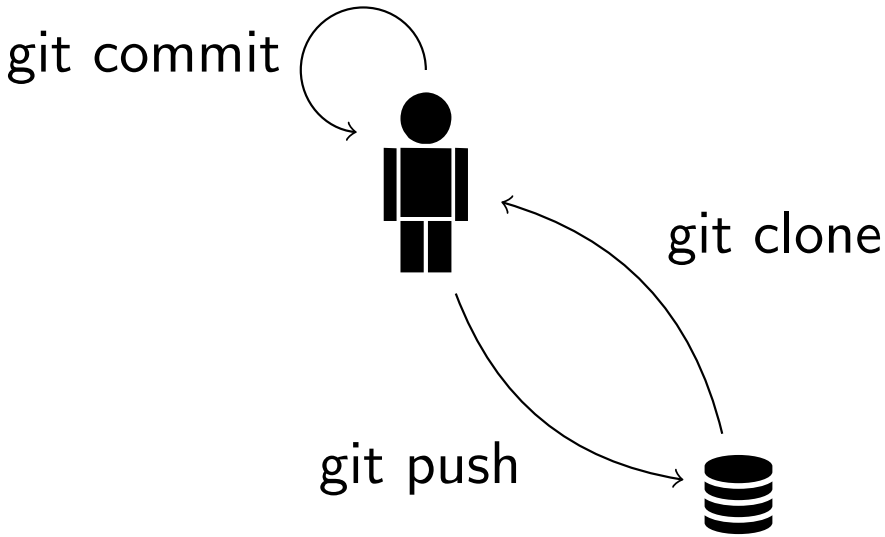
git-push

- Updates remote/server with the local changes
  - Changes will be visible to everyone

git-push

- Updates remote/server with the local changes
  - Changes will be visible to everyone
  - More robust to storage faults (if stored remotely)

git-push cont.



git-fetch & git-pull

- *git-fetch* fetches changes from the remote

git-fetch & git-pull

- *git-fetch* fetches changes from the remote

  git fetch

git-fetch & git-pull

- *git-fetch* fetches changes from the remote

  git fetch

- *git-pull* fetches changes from the remote and integrates the changes

git-fetch & git-pull

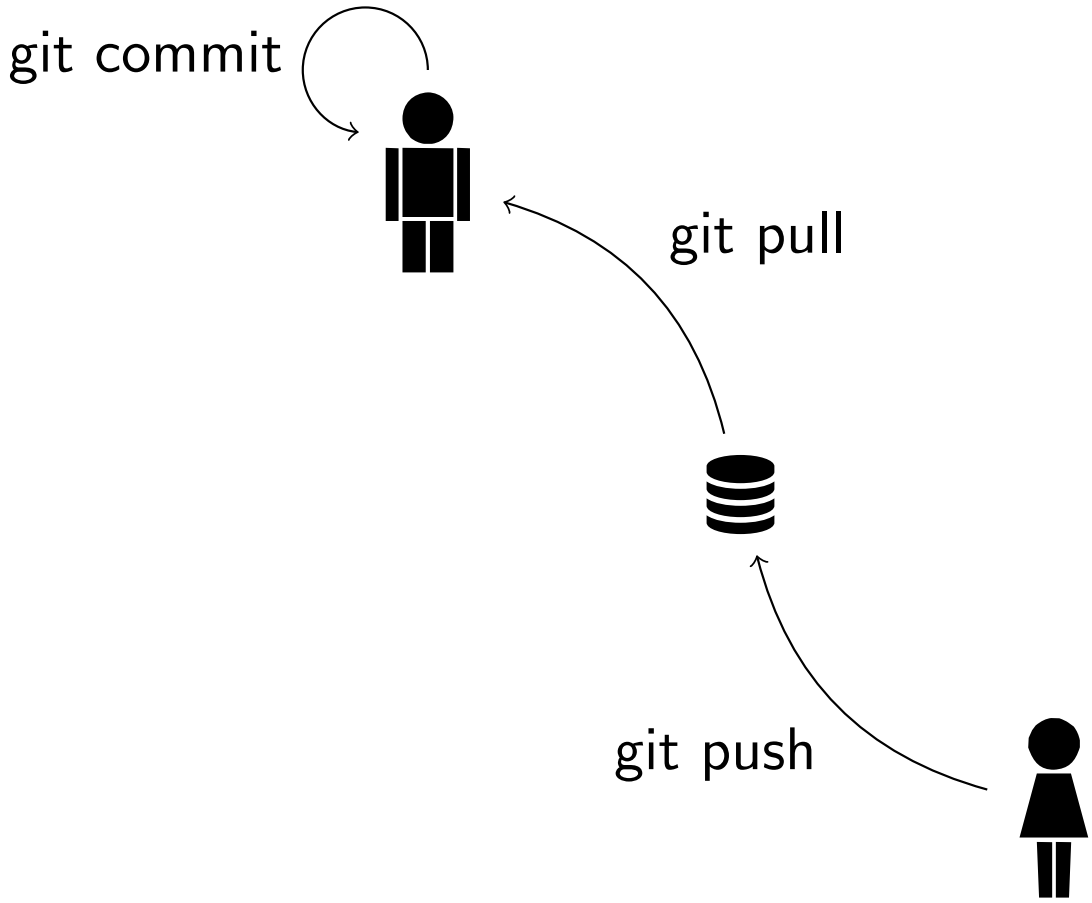
- *git-fetch* fetches changes from the remote

  git fetch

- *git-pull* fetches changes from the remote and integrates the changes

  git pull

git-fetch & git-pull cont.



git-status & .gitignore file

- *git-status* shows current state of the repository

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)
- A way to ignore unwanted files: .gitignore

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)
- A way to ignore unwanted files: .gitignore
- Uses pattern matching to filter out files to be ignored

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)
- A way to ignore unwanted files: .gitignore
- Uses pattern matching to filter out files to be ignored
  - filter build/Debug folder: “build/” or “Debug/”

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)
- A way to ignore unwanted files: .gitignore
- Uses pattern matching to filter out files to be ignored
  - filter build/Debug folder: "build/" or "Debug/"
  - filter individual files: simply write file name

git-status & .gitignore file

- *git-status* shows current state of the repository
  - Staged and unstaged changes
    git status <a-path-in-repository>
- Compilation spits out a lot of intermediate files
  - Executables/binaries
  - Debug files
  - Raw resources (text, image, etc.)
- A way to ignore unwanted files: .gitignore
- Uses pattern matching to filter out files to be ignored
  - filter build/Debug folder: "build/" or "Debug/"
  - filter individual files: simply write file name
  - filter .txt files: "*.txt"

git-log & git-diff

- *git-log* shows the commit history

  git log

- *git-diff* shows the current changes (the "diff")

  git diff

Time for a small demo.
Let's create a git repo in Bitbucket from scratch.

Commit atomic changes

```
std::string find_surname(int id)
{
    std::map<int, std::string>& data = get_data();
    return data[id];
}
```

Commit atomic changes

```
std::string get_surname(int id) // 1 - change the name
{
    // 2 - fix a potential bug
    const std::map<int, std::string>& data = get_data();
    return data.at(id);
}
```

Commit early & commit often

- Related to atomic changes

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small
- Small changes can be committed frequently

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small
- Small changes can be committed frequently
- Inconvenient but useful. *How?*

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small
- Small changes can be committed frequently
- Inconvenient but useful. *How?*
  - Easily roll back on experimental changes

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small
- Small changes can be committed frequently
- Inconvenient but useful. *How?*
  - Easily roll back on experimental changes
  - Easier to develop two or more “features” in parallel

Commit early & commit often

- Related to atomic changes
- Atomic changes tend to be small
- Small changes can be committed frequently
- Inconvenient but useful. *How?*
  - Easily roll back on experimental changes
  - Easier to develop two or more “features” in parallel
  - Early feedback (extremely important)

Importance of commit messages

- Why is this important?

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did
  - People who are alien to code base might need historical context

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did
  - People who are alien to code base might need historical context
- How to improve?

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did
  - People who are alien to code base might need historical context
- How to improve?
  - Be grammatically correct

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did
  - People who are alien to code base might need historical context
- How to improve?
  - Be grammatically correct
  - Do not be afraid of writing long texts when needed

Importance of commit messages

- Why is this important?
  - Large code changes can be summed up clearly
  - You may forget what your code did
  - People who are alien to code base might need historical context
- How to improve?
  - Be grammatically correct
  - Do not be afraid of writing long texts when needed
  - Express “why” (intent) and not “how” (most common problem)
    - Do not repeat what the code already says
    - Changes to code already show the “how” part

Importance of commit messages cont.

```
std::string find_surname(int id)
{
    const std::map<int, std::string>& data = get_data();
    return data.at(id);
}
```

"Use const & and .at operator in get_surname method"

Importance of commit messages cont.

```
std::string find_surname(int id)
{
    const std::map<int, std::string>& data = get_data();
    return data.at(id);
}
```

"Use const & and .at operator in get_surname method"

"Avoid inserting new elements to data when id is not found"

Summary

- Vital to software development for coordination and change tracking

Summary

- Vital to software development for coordination and change tracking

  git clone

  git add ...

  git commit

  git push

Summary

- Vital to software development for coordination and change tracking

  git clone

  git add ...

  git commit

  git push
- Commit small changes

Summary

- Vital to software development for coordination and change tracking

  git clone

  git add ...

  git commit

  git push
- Commit small changes
- Express “why” in commit messages

Some resources

git help <command-name>

https://git-scm.com/doc

https://gitexplorer.com

Closure

Thanks!
Questions?