CS 563: Software Maintenance And Evolution

Software Change Management

Oregon State University, Spring 2024

Today’s plan

- Learn about
  - Software change management process
  - Various kinds of version control systems
  - Overview the basics of git
  - Touch some intermediate git topics
  - Clear up common points of confusion
    - Branch vs Fork?
    - Merge vs Pull Request?
    - Pull vs Fetch?
    - Fork vs Clone?
- BRING A LAPTOP! FOR IN-CLASS EXERCISES

Software Change Management Process

Meet Tom,
the developer

Software Change Management Process

Meet Tom,
the developer

When Tom writes his code alone:

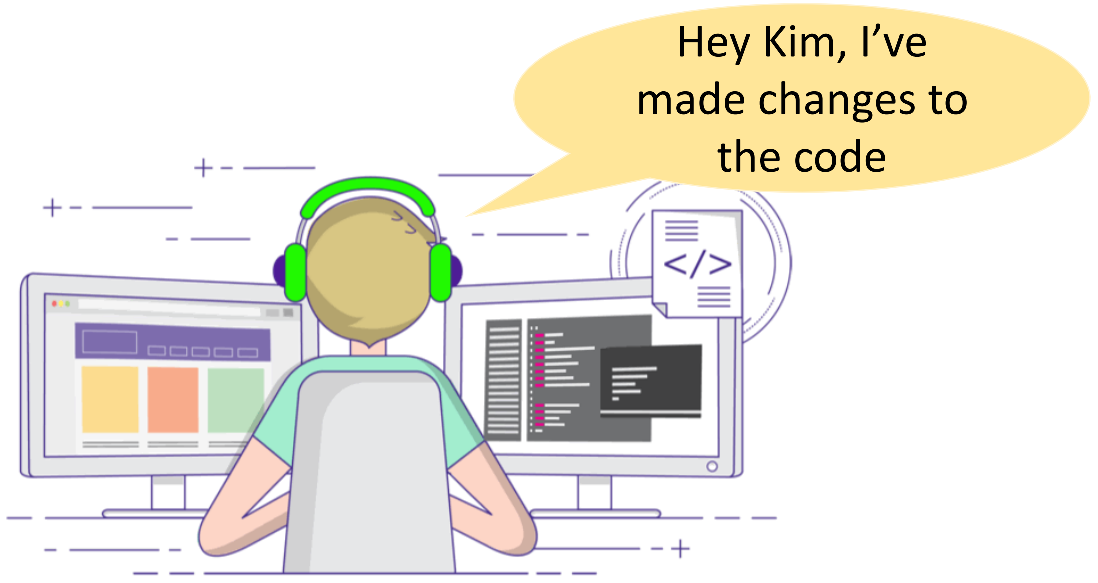
- He knows what code changes he made
- He doesn't have to worry about where the files are kept because he put them there
- He is the sole owner of the code, and he knows every last detail of the code

Software Change Management Process

Meet Tom,
the developer

What if we add
Kim to this
workspace

Software Change Management Process



Meet Tom, the developer

What if we add Kim to this workspace

Software Change Management Process

Meet Tom, the developer

What if we add Kim to this workspace

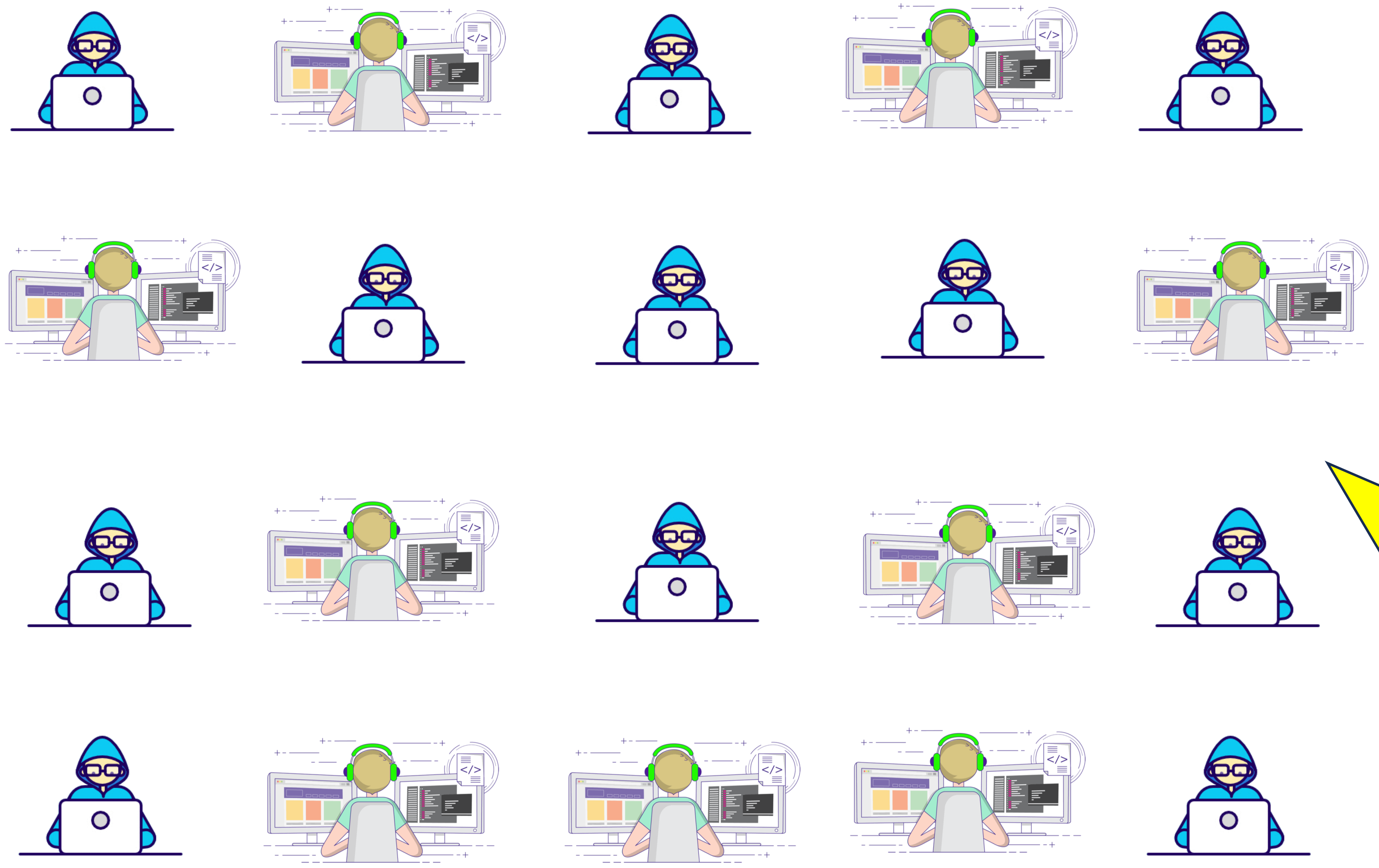
Software Change Management Process

Meet Tom, the developer

What if we add Kim to this workspace

Simple! Isn't it?

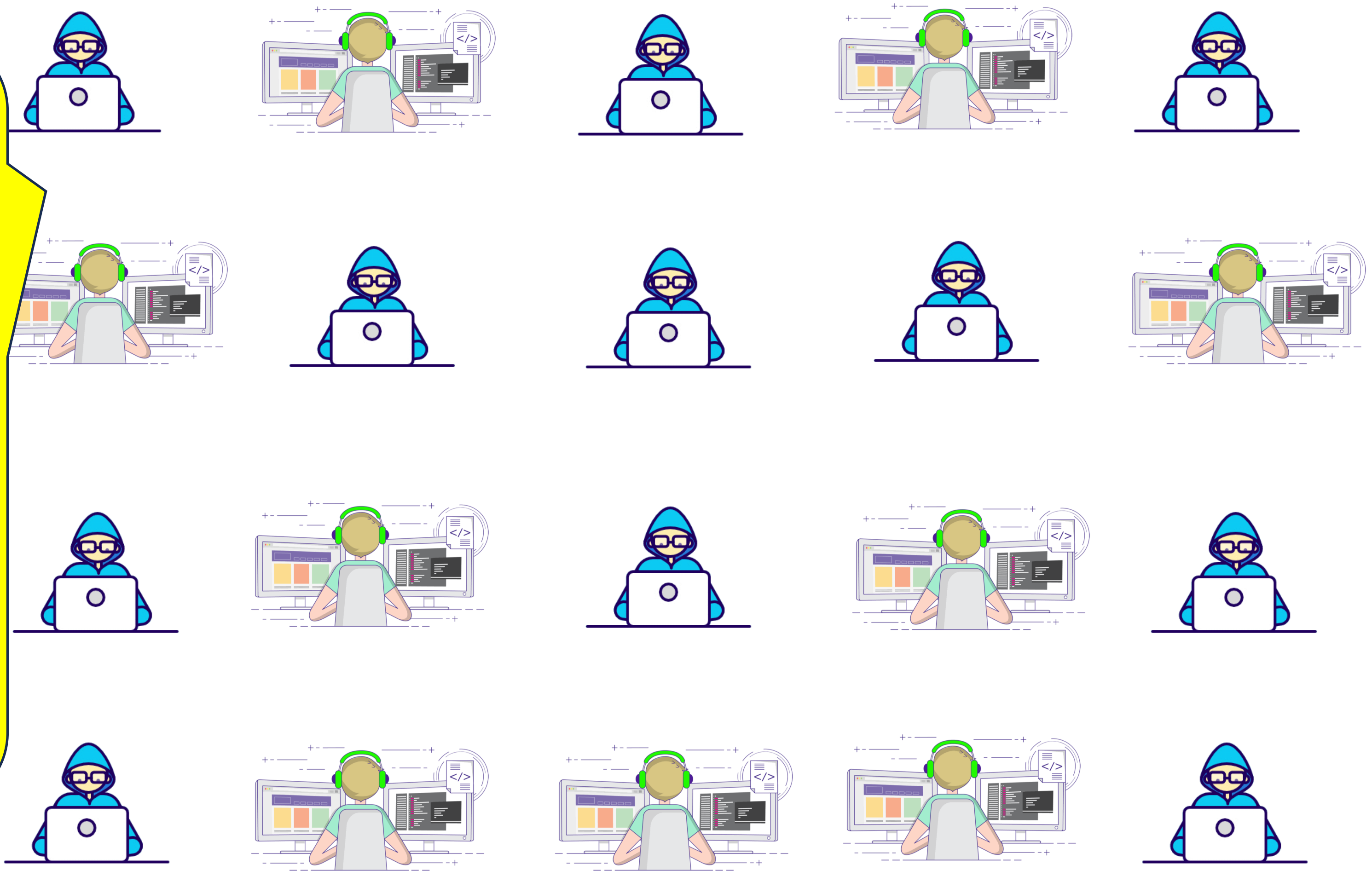
Software Change Management Process



Not really! Because it won't work in huge organizations!

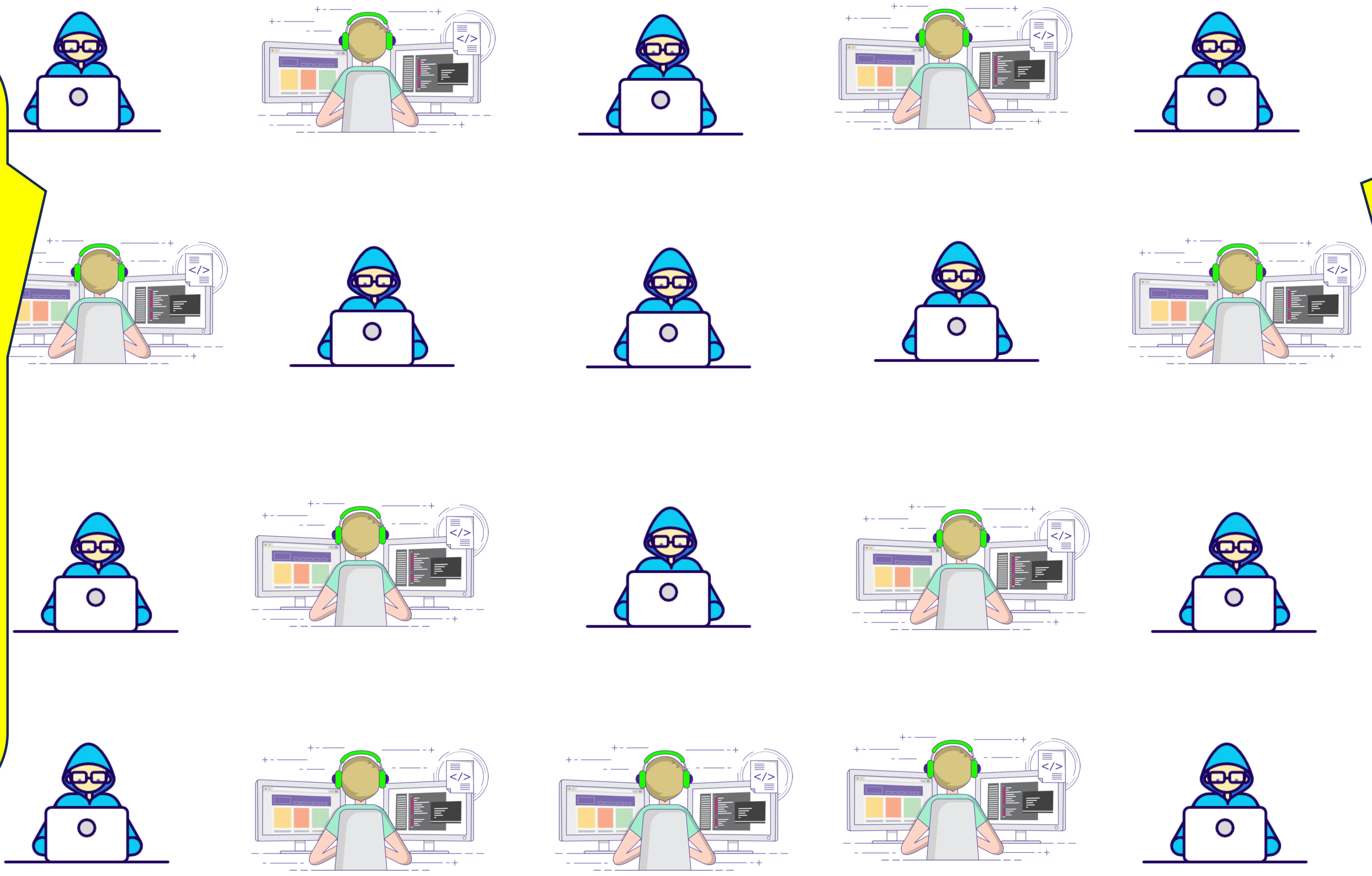
Software Change Management Process

How can we ensure everyone is working without any conflicts in code while changes are made by different developers?



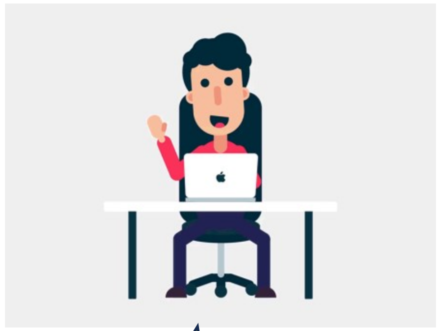
Software Change Management Process

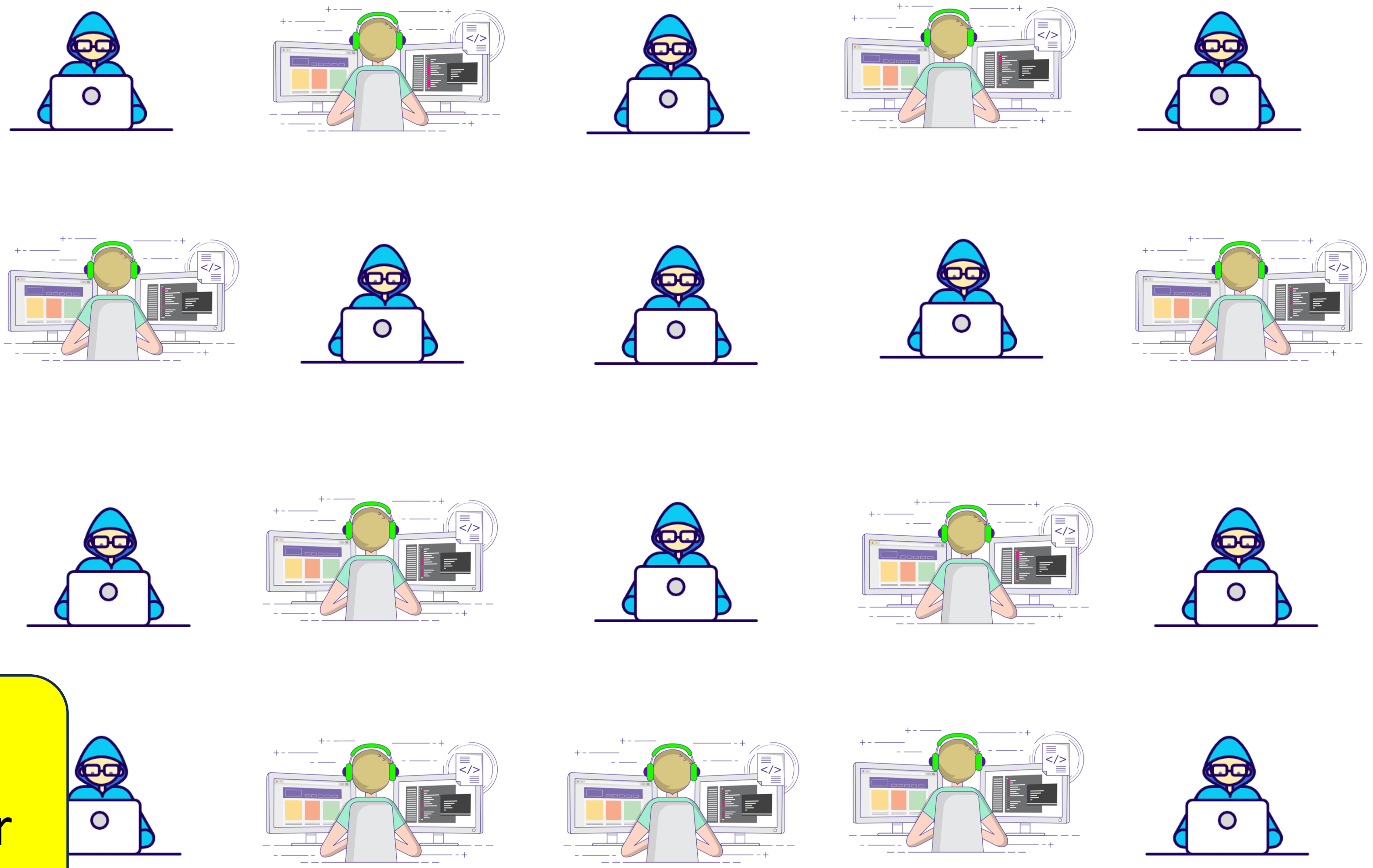
How can we ensure everyone is working without any conflicts in code while changes are made by different developers?



How can we ensure that there won't be any communication gaps when code changes are made?

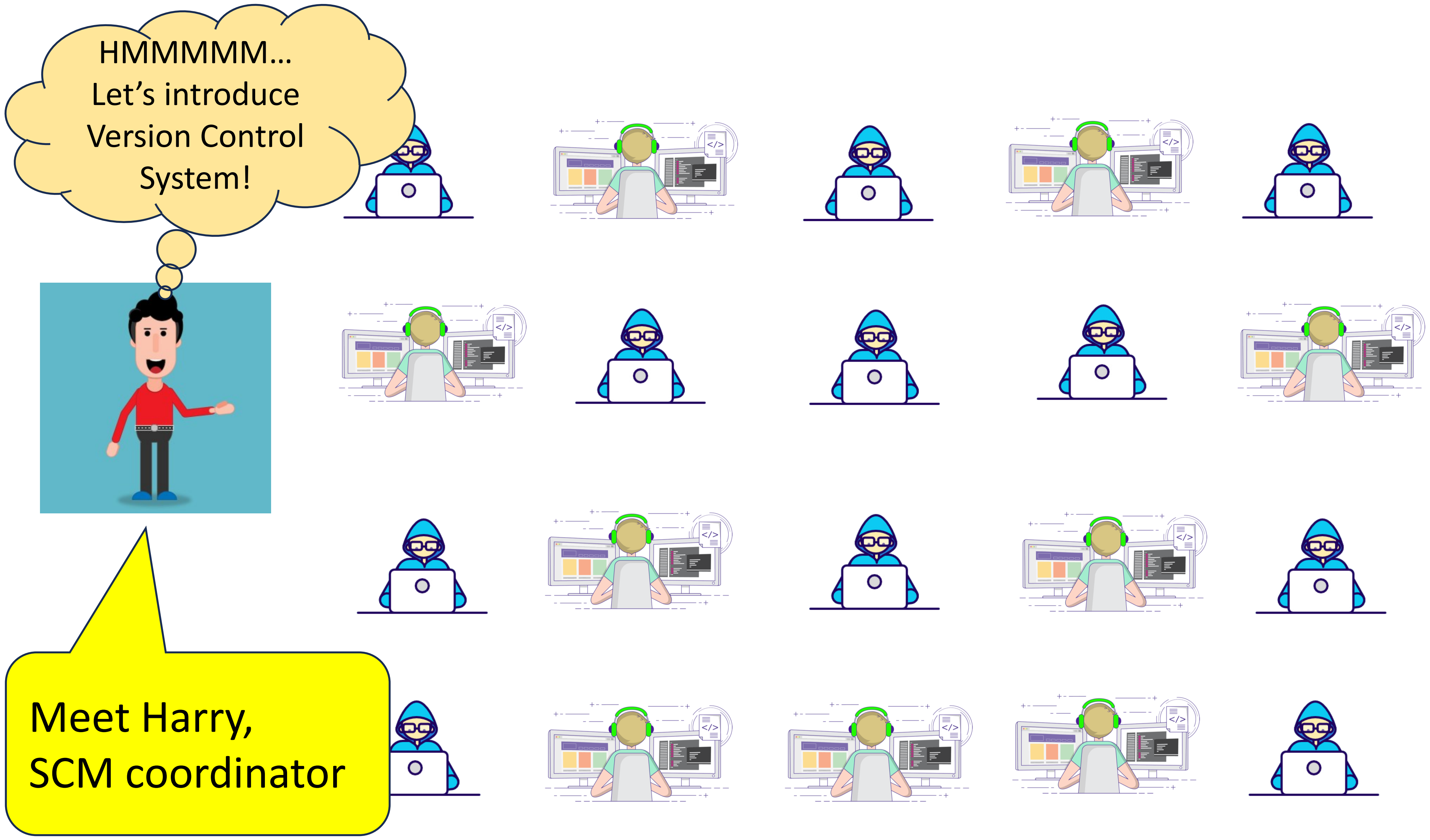
Software Change Management Process



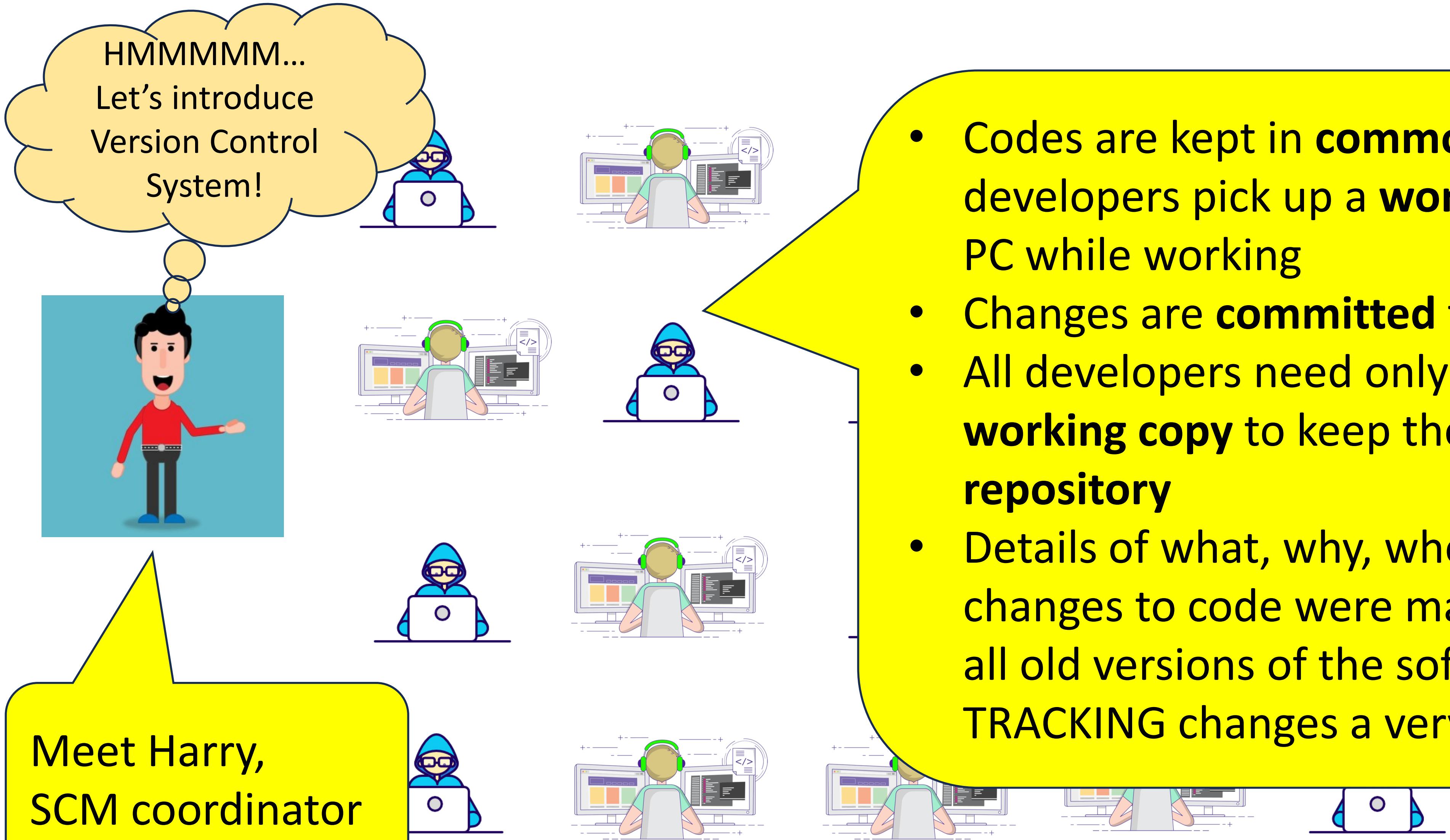


Meet Harry,
SCM coordinator

Software Change Management Process


HMMMMM… Let’s introduce Version Control System!
Meet Harry, SCM coordinator

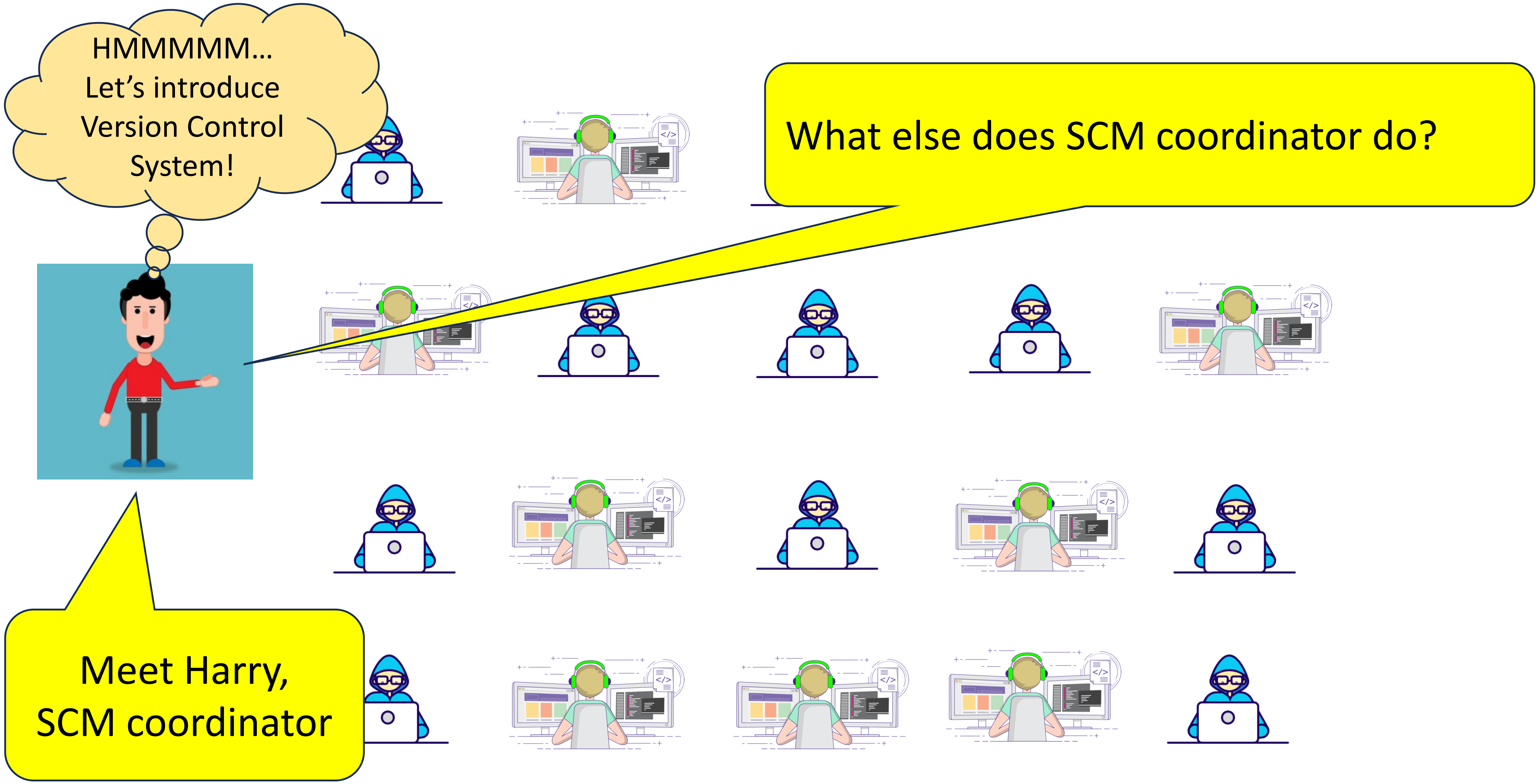
Software Change Management Process



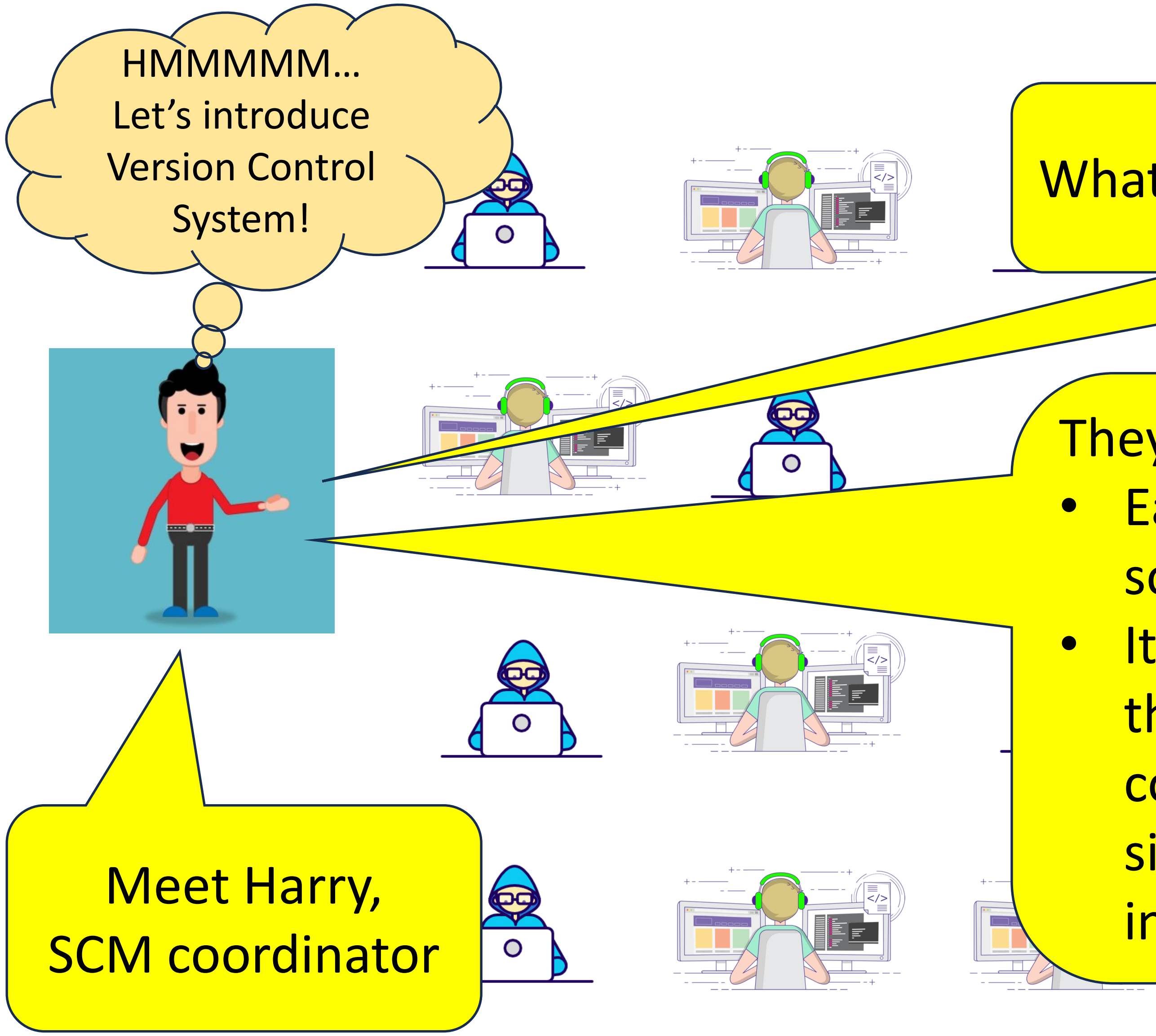
- Codes are kept in **common repository** & developers pick up a **working copy** to their PC while working
- Changes are **committed** to the repository
- All developers need only **UPDATE** their **working copy** to keep them in **sync** with the **repository**
- Details of what, why, when, and by whom changes to code were made. This applies to all old versions of the software which makes TRACKING changes a very simple task.

Meet Harry,
SCM coordinator

Software Change Management Process


HMMMMM… Let’s introduce Version Control System!
What else does SCM coordinator do?
Meet Harry, SCM coordinator

Software Change Management Process



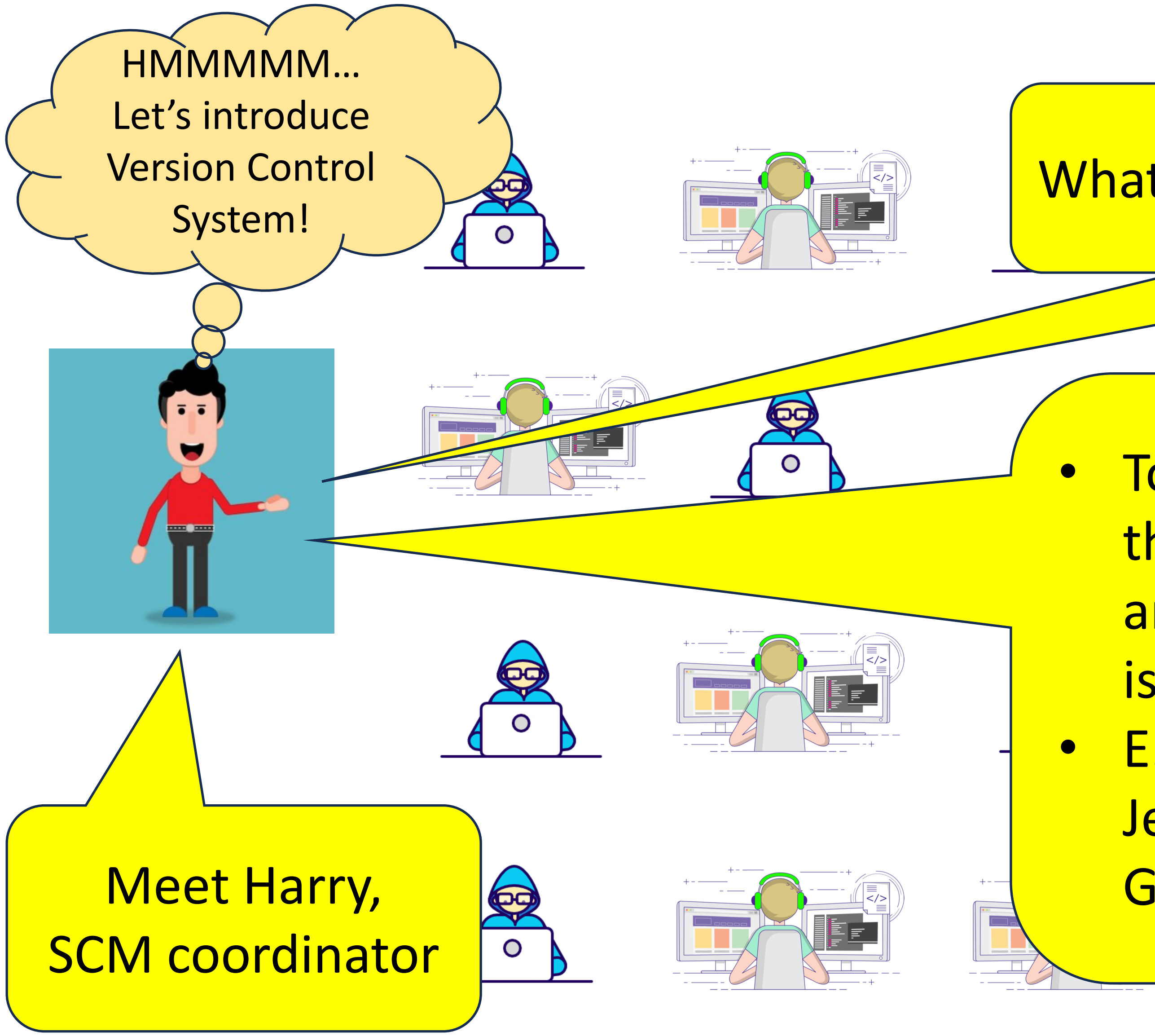
Meet Harry, SCM coordinator

What else does SCM coordinator do?

They also take care if builds.

- Each developer works on different software components.
- It is SCM coordinator's job to ensure that all the codes written are compiled and integrated into a single package, which is deployed into the production server.

Software Change Management Process

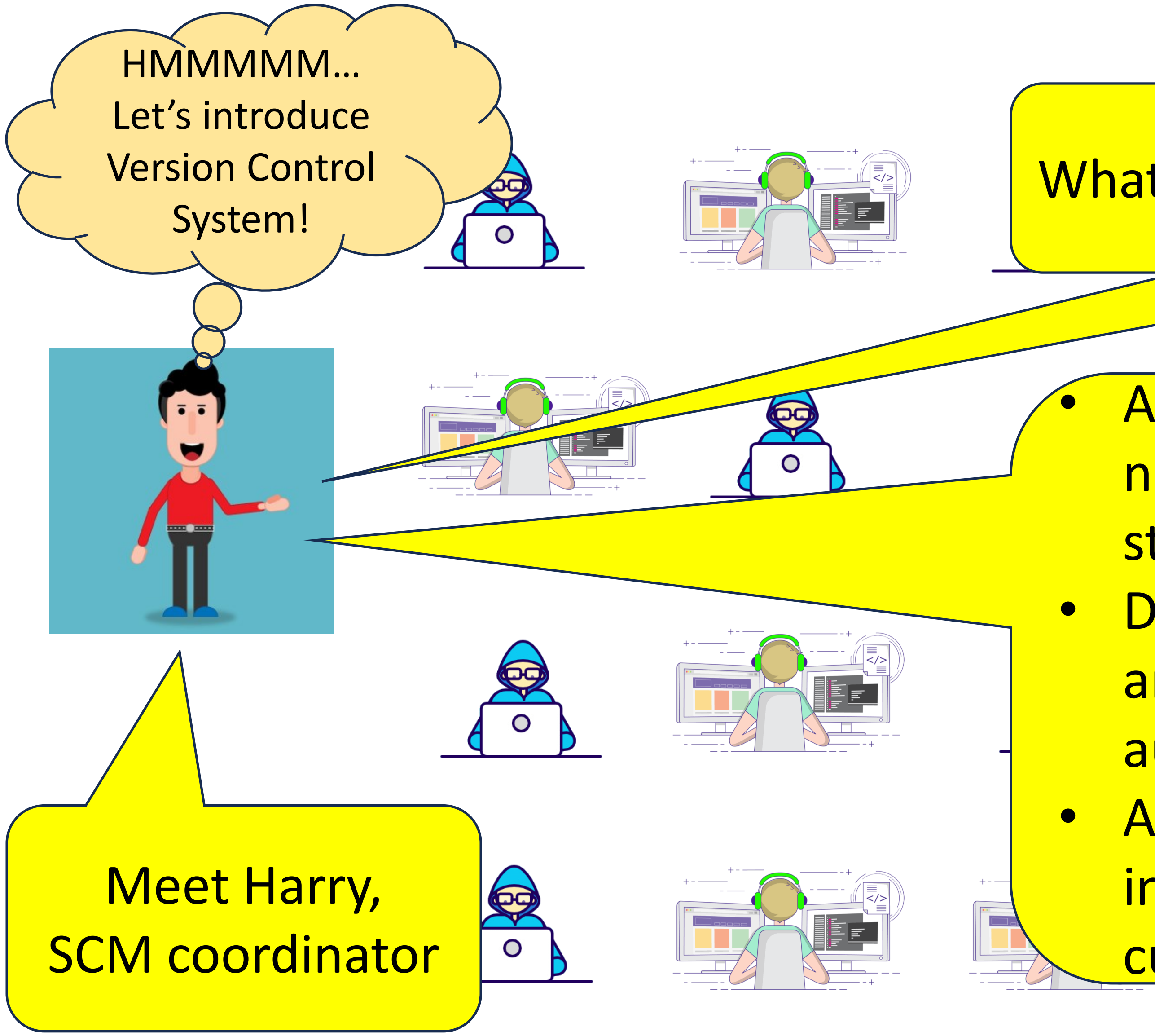


Meet Harry, SCM coordinator

What else does SCM coordinator do?

- To take care of builds automatically they make use of CI/CD pipelines and write scripts to ensure that code is correctly compiled and packaged.
- E.g., tools: Maven, Ant, Build Forge, Jenkins, BAMBOO, CruiseControl, Github Actions, etc.

Software Change Management Process

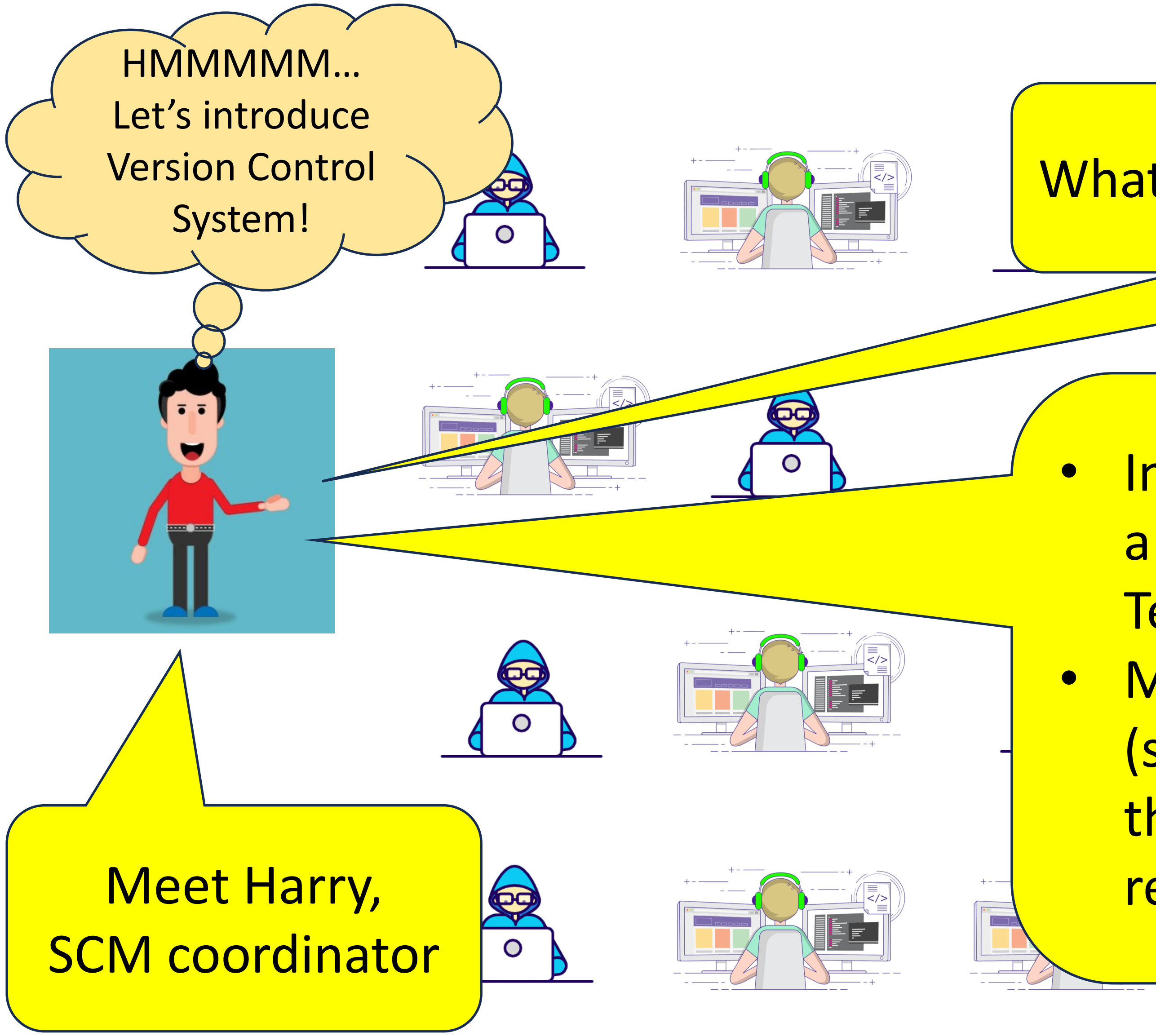


Meet Harry,
SCM coordinator

What else does SCM coordinator do?

- Additionally, they prepare release notes, schedule deployment dates, status update for releases.
- Documentation of all the changes and releases plays a vital role during audits.
- Also spends time producing innovative ways of automating current processes

Software Change Management Process


HMMMMM...
Let's introduce Version Control System!
Meet Harry, SCM coordinator

What else does SCM coordinator do?

- In fact, the SCM coordinator acts as a link between Development, Testing, and the Production Servers.
- Managing the Environments (software and hardware that host the system) is another important responsibility assigned to them.

Version Control Systems

A version control system:

- Records changes to a file set or code base over time, making it easy to review or revert to specific versions later

Version Control Systems

A version control system:

- Records changes to a file set or code base over time, making it easy to review or revert to specific versions later
- Enables multiple people to simultaneously work on a single project

Version Control Systems

A version control system:

- Records changes to a file set or code base over time, making it easy to review or revert to specific versions later
- Enables multiple people to simultaneously work on a single project
- Enables one person to use multiple computers to work on a single project (super useful even if you are working by yourself!)

Version Control Systems

A version control system:

- Records changes to a file set or code base over time, making it easy to review or revert to specific versions later
- Enables multiple people to simultaneously work on a single project
- Enables one person to use multiple computers to work on a single project (super useful even if you are working by yourself!)
- Integrates work done simultaneously by different teams / team members

Version Control Systems

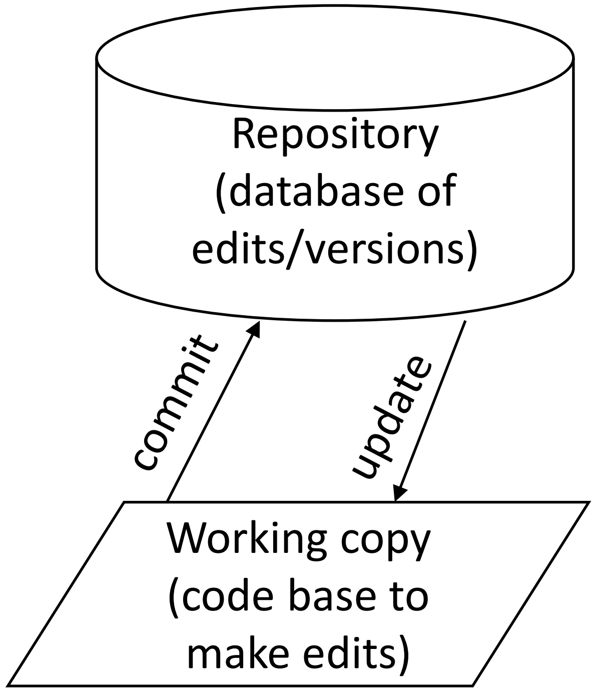
A version control system:

- Records changes to a file set or code base over time, making it easy to review or revert to specific versions later
- Enables multiple people to simultaneously work on a single project
- Enables one person to use multiple computers to work on a single project (super useful even if you are working by yourself!)
- Integrates work done simultaneously by different teams / team members
- Gives access to historical versions of your project

Repository and working copy

A version control system uses:

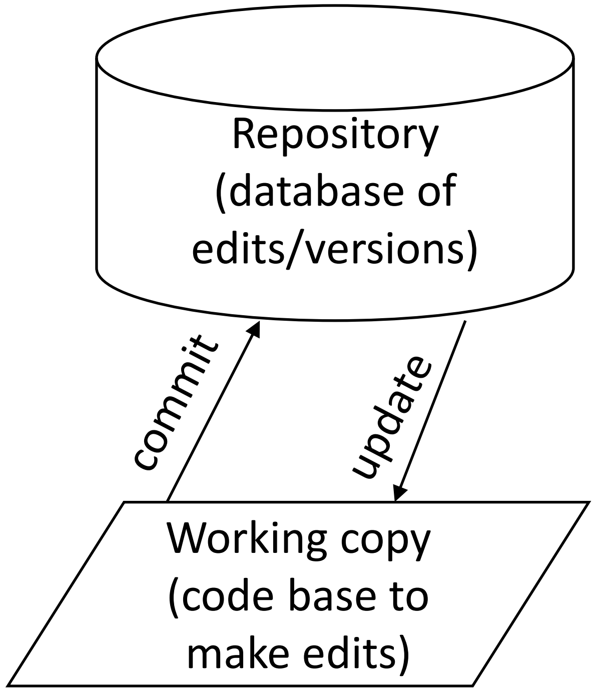
- *Repository*: A database of changes
- Working copy (aka *checkout*): The code base where you do your work



Repository and working copy

A version control system uses:

- *Repository*: A database of changes
- Working copy (aka *checkout*): The code base where you do your work

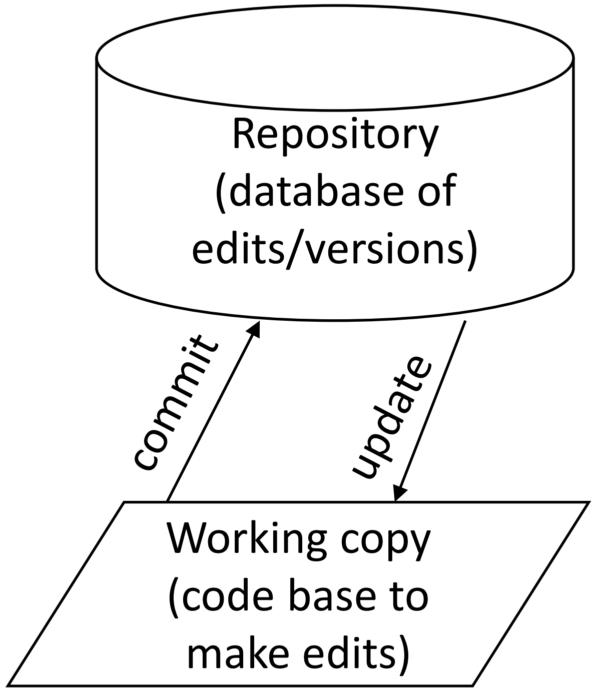

Repository (database of edits/versions)
commit
update
Working copy (code base to make edits)

When you are happy with your edits in your **working copy**, you **commit** your changes to a **repository**

Repository and working copy

A version control system uses:

- *Repository*: A database of changes
- Working copy (aka *checkout*): The code base where you do your work


Repository (database of edits/versions)
commit
update
Working copy (code base to make edits)

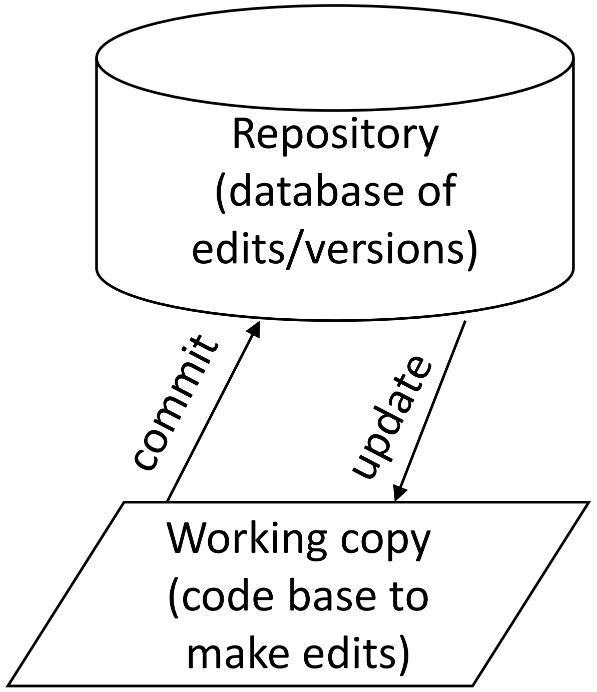
When you are happy with your edits in your **working copy**, you **commit** your changes to a **repository**

It is possible for the **repository** to contain edits that have not yet been applied to your **working copy**

Repository and working copy

A version control system uses:

- *Repository*: A database of changes
- Working copy (aka *checkout*): The code base where you do your work


Repository (database of edits/versions)
commit
update
Working copy (code base to make edits)

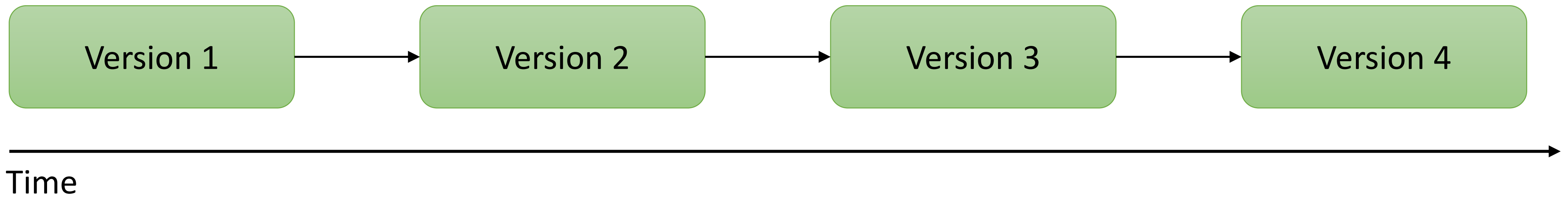
When you are happy with your edits in your **working copy**, you **commit** your changes to a **repository**

It is possible for the **repository** to contain edits that have not yet been applied to your **working copy**

You can **update** your **working copy** to incorporate any new edits or versions that have been added to the **repository**

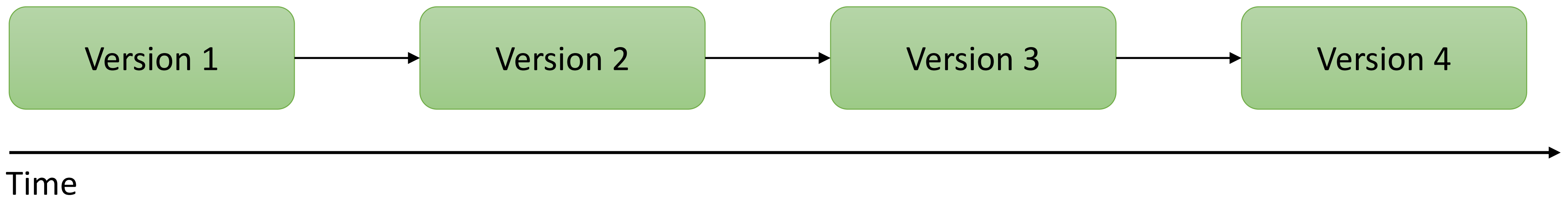
What happens inside a Repository?

- **Simplest case**: the database contains a liner history (each change is made after the previous one)

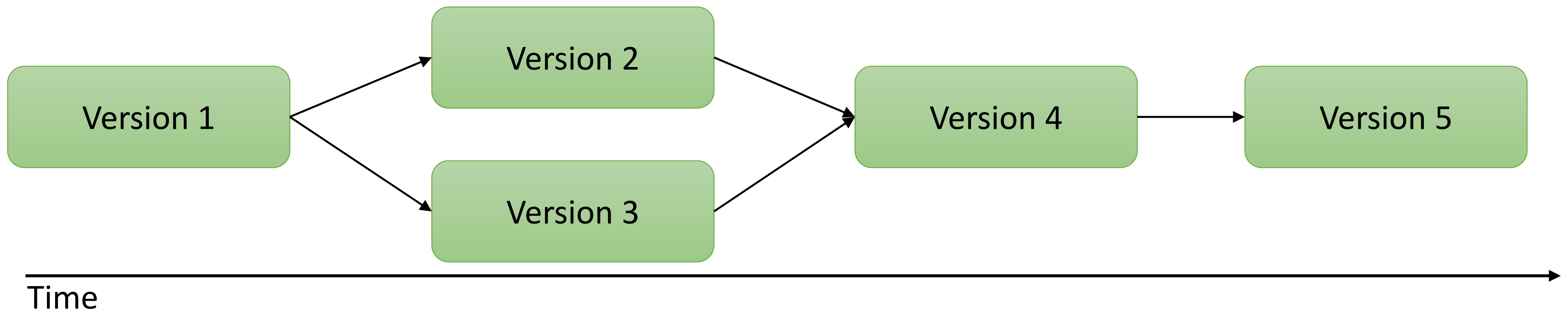


What happens inside a Repository?

- **Simplest case**: the database contains a liner history (each change is made after the previous one)



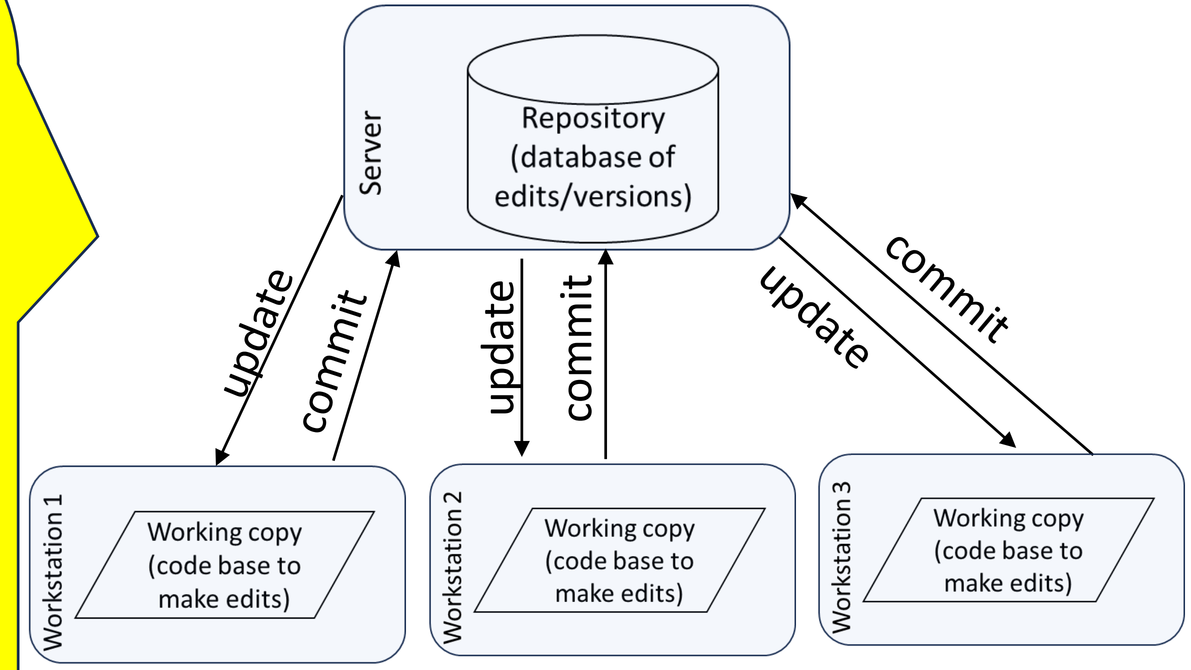
- **Branching:** different users made edits simultaneously and the version history splits and then merges again



Version Control System Types

- Each user gets their own **working copy**
- **One central remote repository**
- You make changes in your **working copy**, ***update*** your **working copy** with latest version of **remote repository**, and ***commit*** your changes.
- Others can now ***update*** their **working copy** to see your changes in their **working copy**

Centralized Version Control System

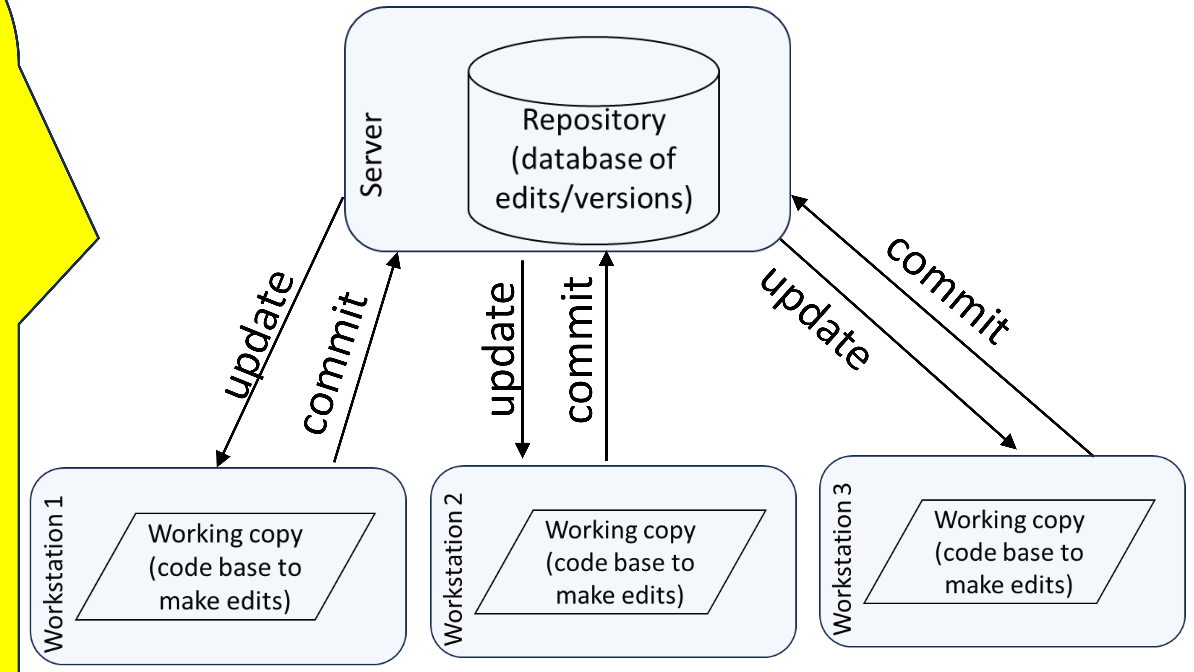

Server
Repository (database of edits/versions)
update
commit
update
commit
update
commit
Workstation 1
Working copy (code base to make edits)
Workstation 2
Working copy (code base to make edits)
Workstation 3
Working copy (code base to make edits)

Version Control System Types

Problems with centralized VCS?

- What if I don’t have a network connection?
- What if I am implementing a big change?
- What if I want to explore project history later?

Centralized Version Control System


Server
Repository (database of edits/versions)
update
commit
update
commit
update
commit
Workstation 1
Working copy (code base to make edits)
Workstation 2
Working copy (code base to make edits)
Workstation 3
Working copy (code base to make edits)

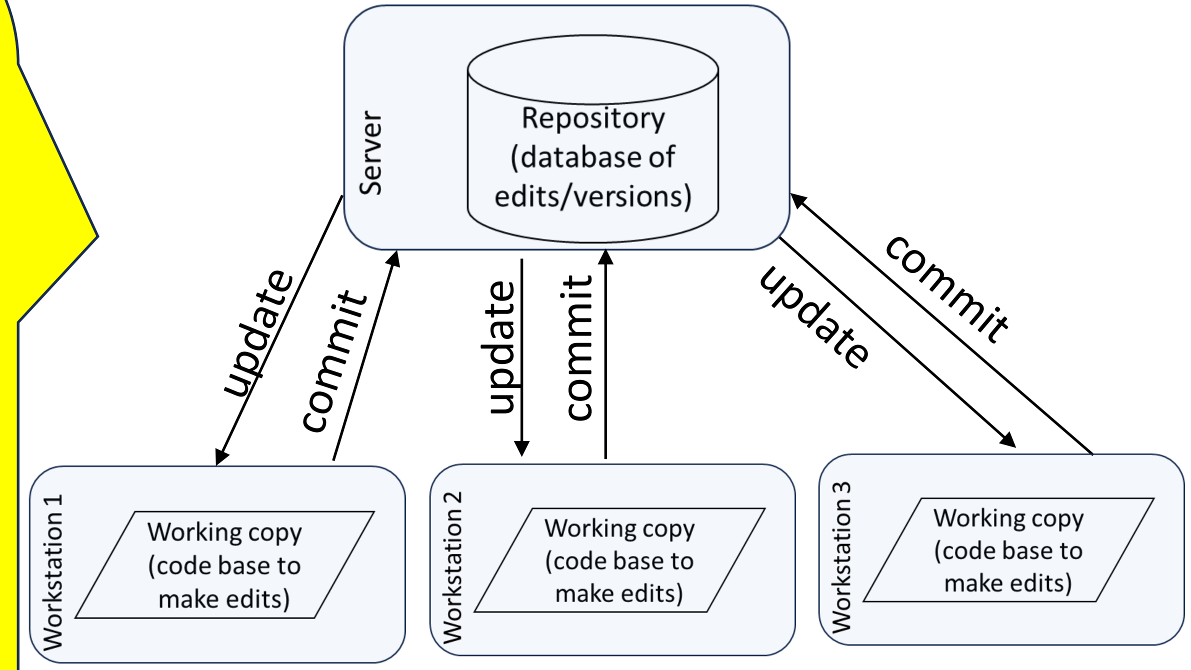
Version Control System Types

Problems with centralized VCS?

- What if I don't have a network connection?
- What if I am implementing a big change?
- What if I want to explore project history later?

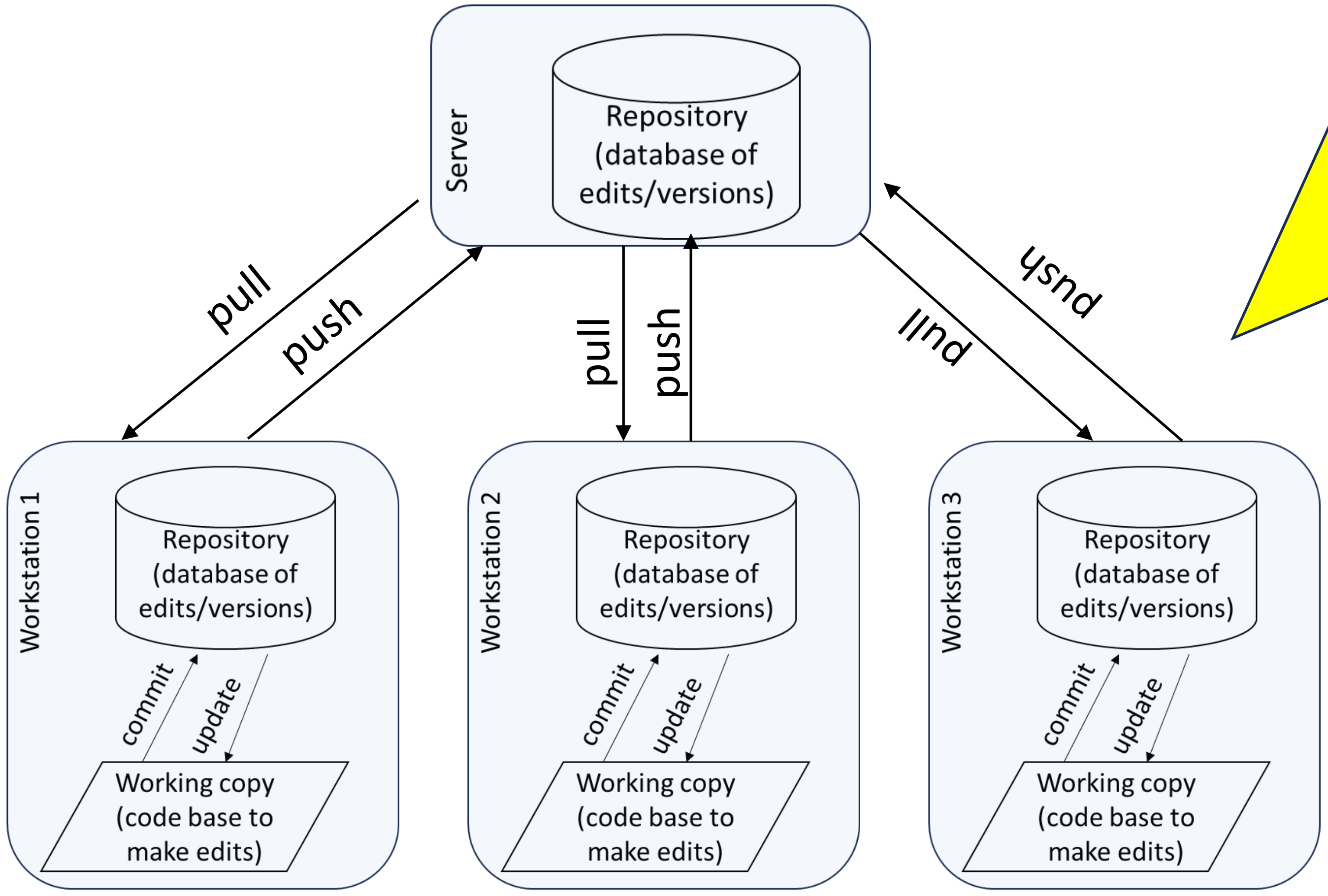
Old model – not used widely anymore

Centralized Version Control System


Server
Repository (database of edits/versions)
update
commit
update
commit
update
commit
Workstation 1
Working copy (code base to make edits)
Workstation 2
Working copy (code base to make edits)
Workstation 3
Working copy (code base to make edits)

Version Control System Types

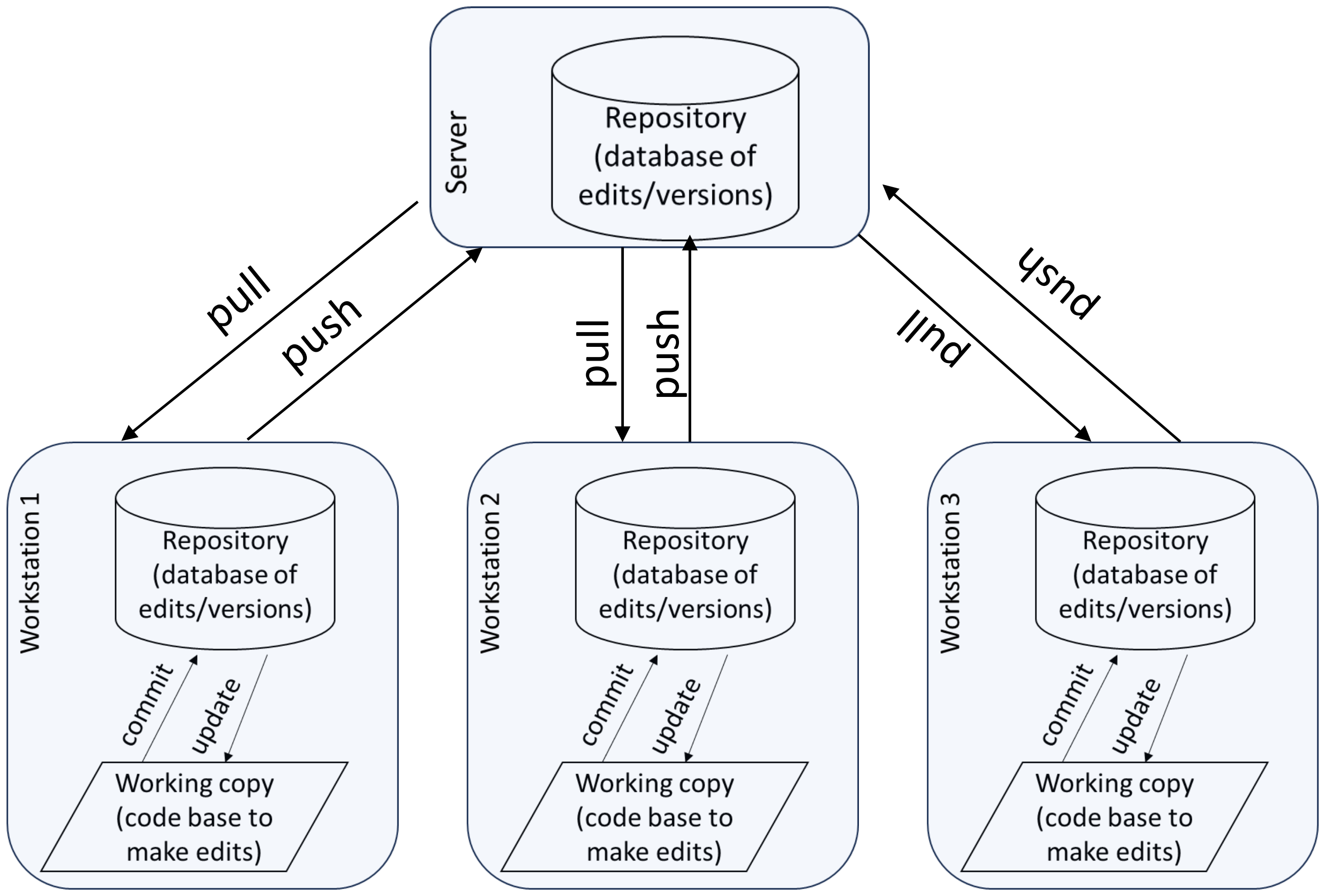
Distributed Version Control System


Server
Repository (database of edits/versions)
pull
push
pull
push
push
pull
Workstation 1
Workstation 2
Workstation 3
Repository (database of edits/versions)
commit
update
Working copy (code base to make edits)

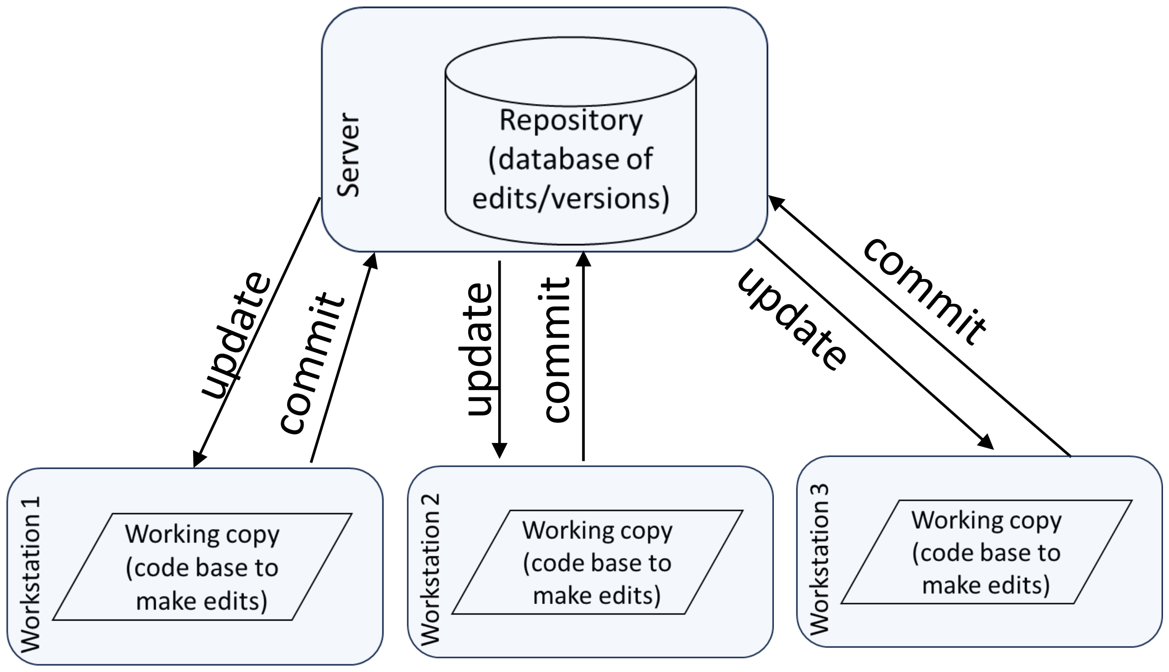
- Each user gets their own **working copy** and **repository**
- **One central remote repository**
- You ***commit*** and ***update*** your local repository and working copy
- Once ready, you ***pull*** the latest version of **remote repository** and ***push*** your changes
- Others can now ***pull*** your changes to their **local repository** and ***update*** their **working copy**

Version Control System Types

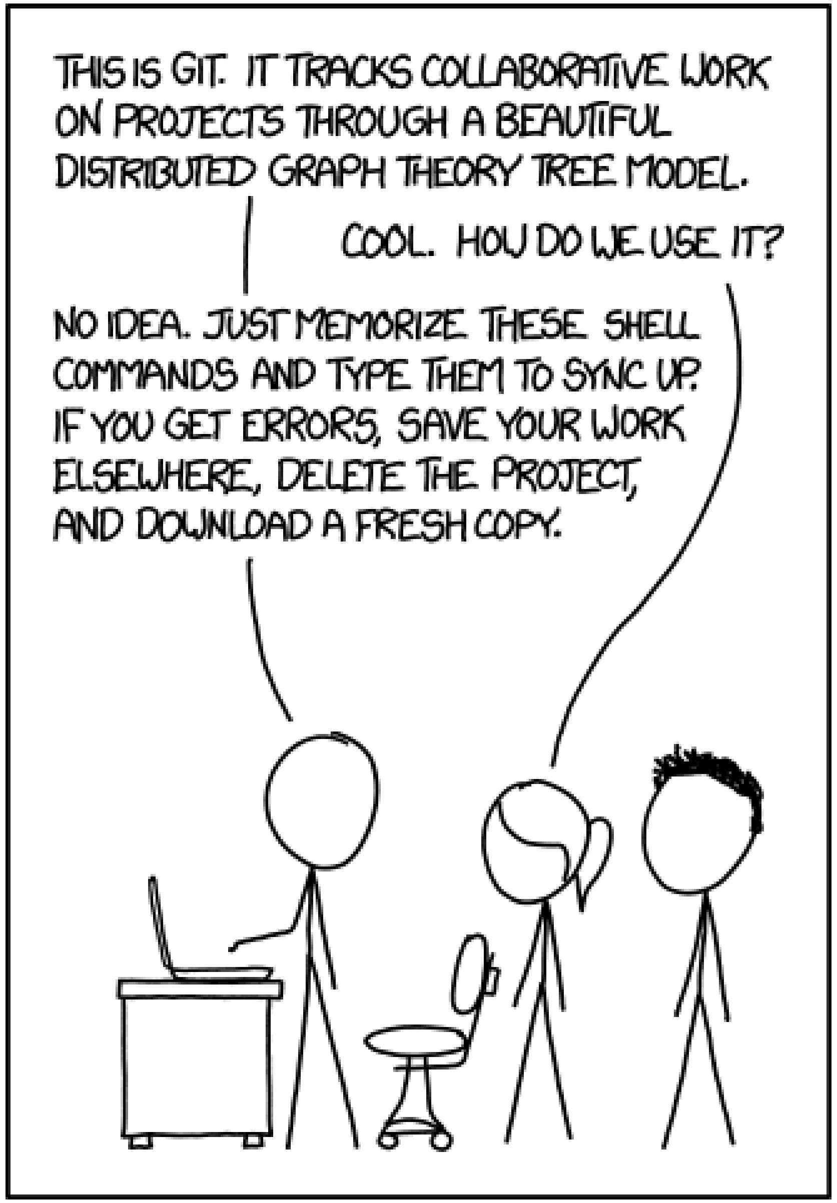
Distributed Version Control System


Server
Repository (database of edits/versions)
pull
push
pull
push
pull
push
Workstation 1
Workstation 2
Workstation 3
Repository (database of edits/versions)
commit
update
Working copy (code base to make edits)

Centralized Version Control System



Popular VCSs: Git (distributed), Mercurial (distributed), Subversion (centralized)


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git’s data model

- *Snapshot*: A version of code base associated with a commit
- Git models the history of collection of files and folders within some top-level directory as a series of snapshots
- In Git terminology:
  - A file is a “**blob**” (represented as bunch of bytes)
  - A folder is a “**tree**”, which maps to other “trees” or “blobs” (so folders can contain sub-folders)
- Modeling history: How should a VCS relate snapshots?

Git's data model

- *Snapshot*: A version of code base associated with a commit
- Git models the history of collection of files and folders within some top-level directory as a series of snapshots
- In Git terminology:
  - A file is a "**blob**" (represented as bunch of bytes)
  - A folder is a "**tree**", which maps to other "trees" or "blobs" (so folders can contain sub-folders)
- Modeling history: How should a VCS relate snapshots?
  - Git uses directed acyclic graph (DAG) of snapshots
  - Git calls these snapshots "commit"s

Git’s data model as pseudo code

```
// a file is a bunch of bytes
type blob = array <bytes>

// a directory contains named files and
directories
type tree = map <string, tree | blob>

// a commit has parents, metadata, and the
top-level tree
type commit = struct {
     parents: array <commit>
     author: string
     message: string
     snapshot:  tree
}
```

Git’s data model as pseudo code

```
// a file is a bunch of bytes
type blob = array <bytes>

// a directory contains named files and
directories
type tree = map <string, tree | blob>

// a commit has parents, metadata, and the
top-level tree
type commit = struct {
     parents: array <commit>
     author: string
     message: string
     snapshot:  tree
}
```

```
type object = blob | tree | commit

objects = map <string, object>

def store(object):
      id = SHA1(object)   // 40 char hexadecimal string
      objects[id] = object

def load(id):
      return objects[id]
```

In-Class Exercise 1: Basic Uses of Git

- Create you account on GitHub and ask me to add your username to https://github.com/CS-563/basic-stats repository

- Form a team of 2 people (at least one of you should have a laptop)

- Download the assignment from https://canvas.oregonstate.edu/courses/1970765/assignments/9661998 and follow the steps.