Some Git Basics

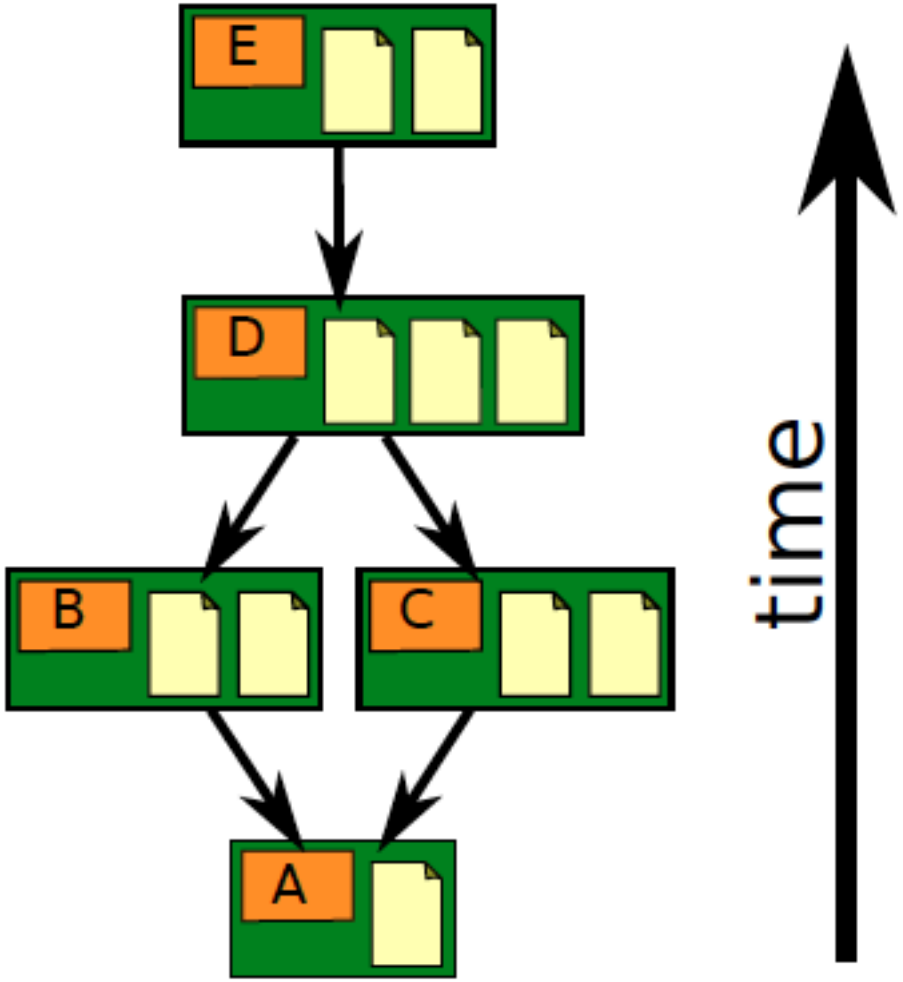
A few useful details
including how to use .gitignore

Git is a Version Control System

A VCS stores files in a repository.

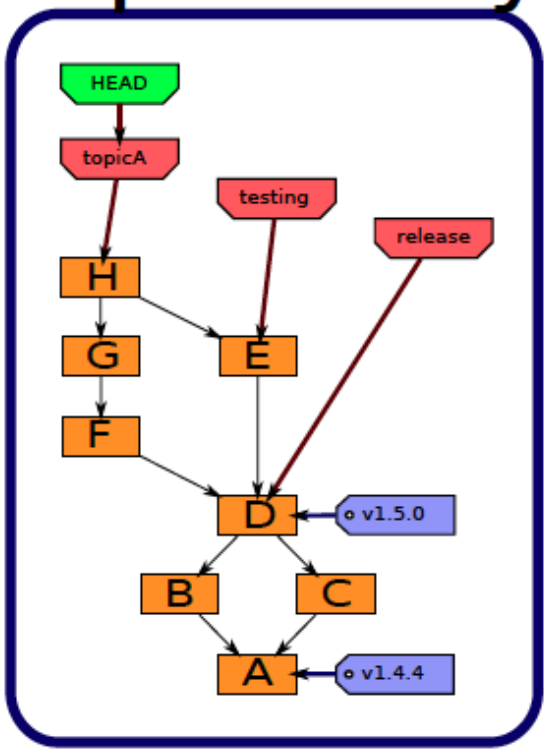
Its keeps a history of all additions & changes, and prevents "old" versions from overwriting newer versions.

Adding new files or updating files is called a "commit".



Git repository

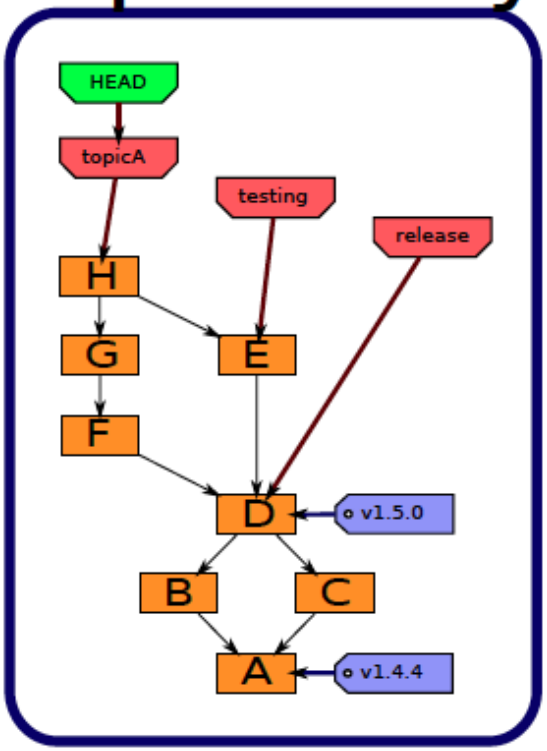
repository



Git repository is usually stored in a subdirectory named **`.git`**

Staging area

repository



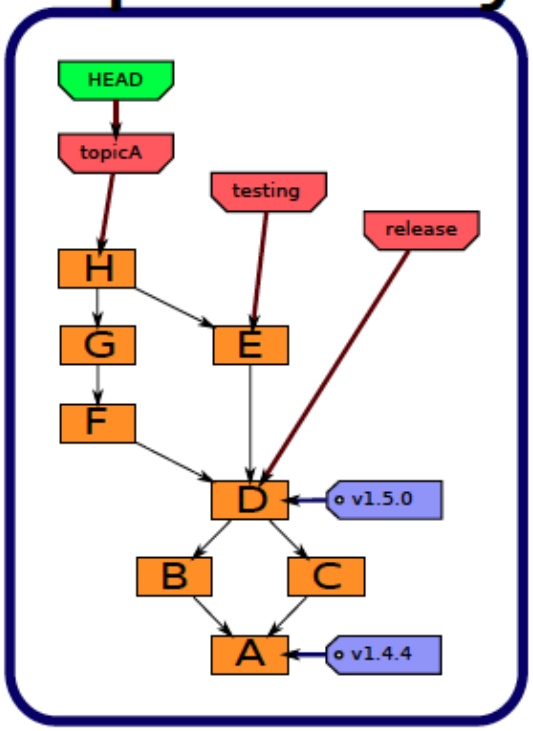
index

```
100644 20b024 0   bar
100644 1d52a6 0   baz
100644 20b024 0   sub/fi
100644 43dbe0 0   sub/foo
```

"Staging area" or "index" is for files and other transactions waiting to be added/deleted/updated in repository.

Your working copy

repository



index

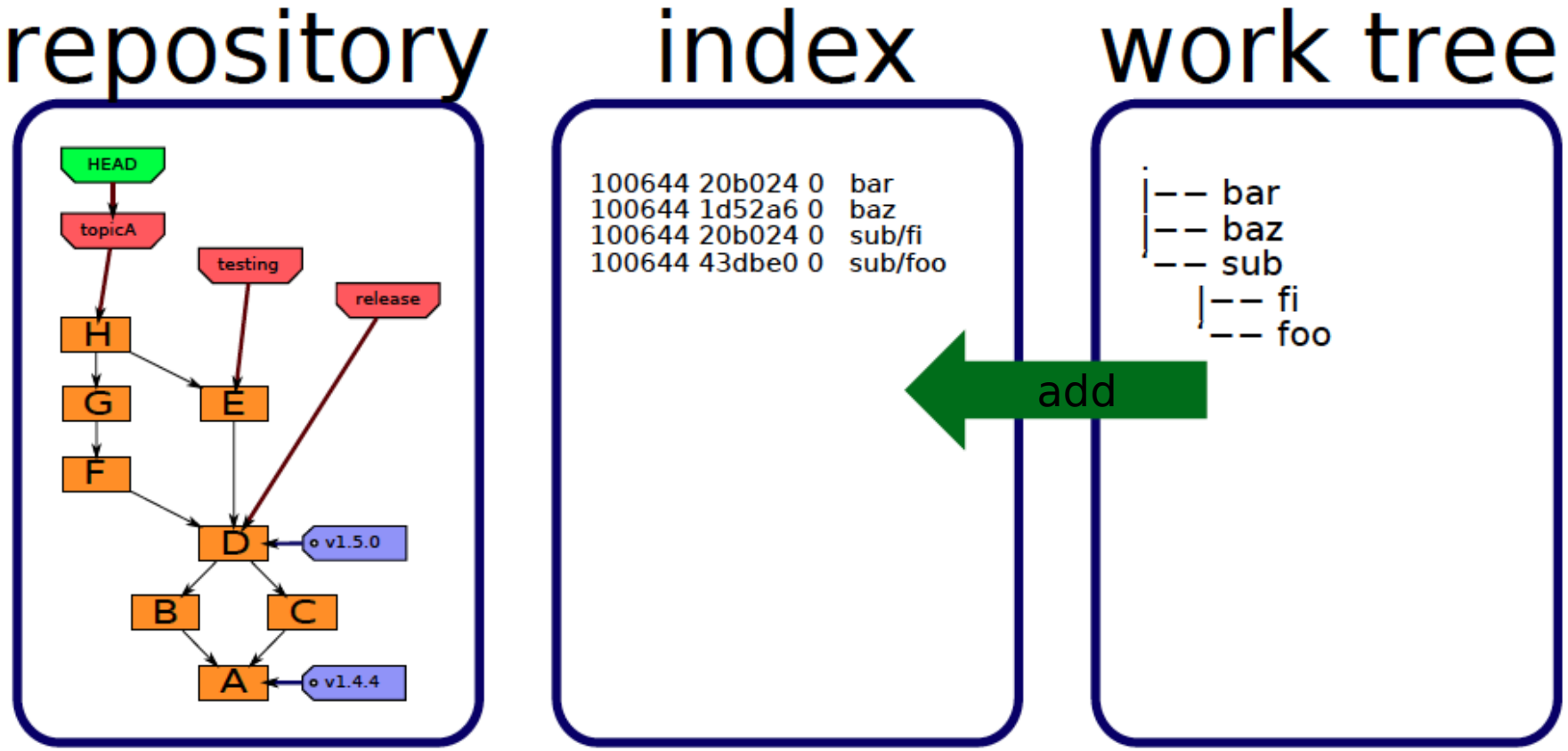
```
100644 20b024 0   bar
100644 1d52a6 0   baz
100644 20b024 0   sub/fi
100644 43dbe0 0   sub/foo
```

work tree

```
.
|-- bar
|-- baz
`-- sub
    |-- fi
    `-- foo
```

Your working tree (normal project) contains files "tracked" by the repository and other files that are not tracked or stored.

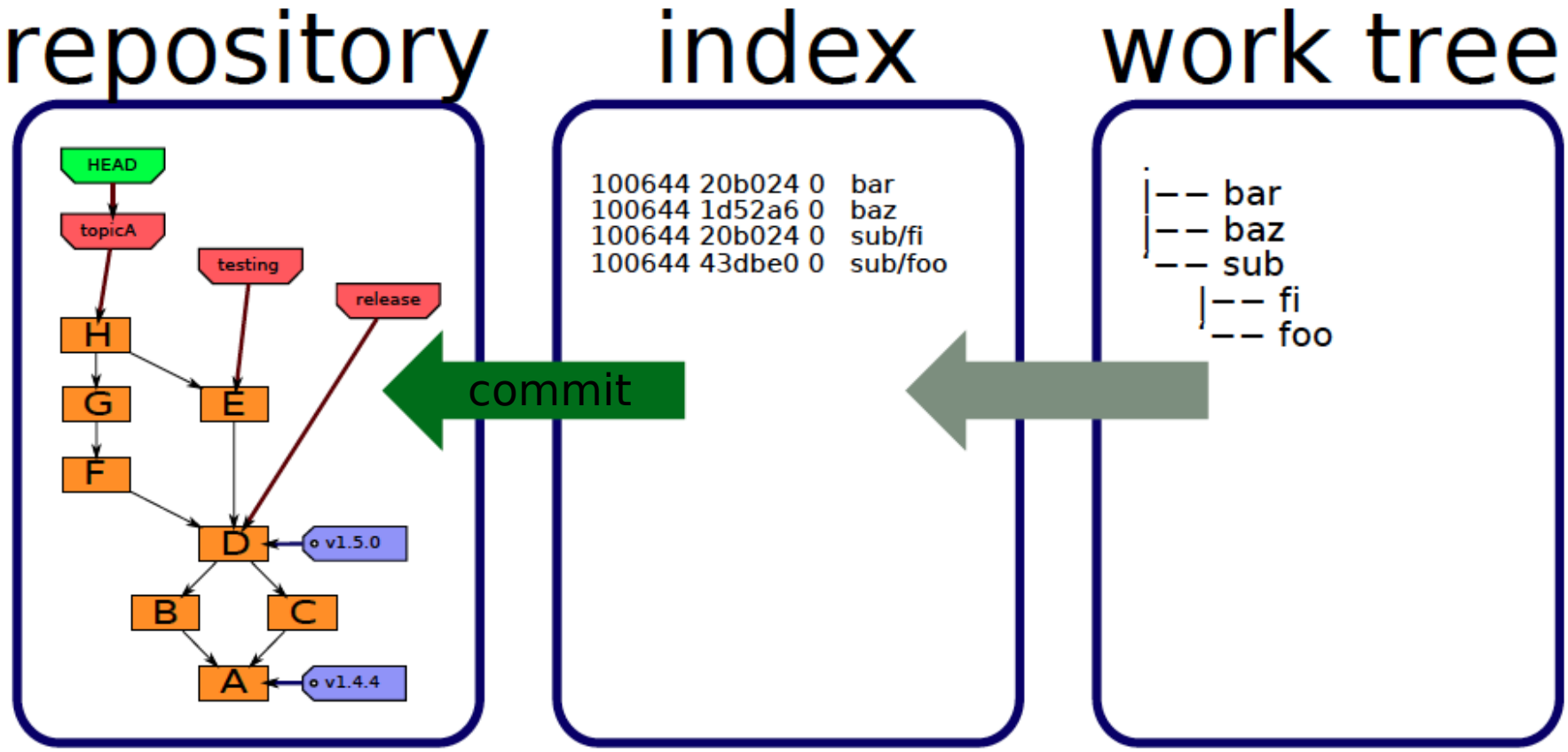
Staging: adding & updating files



There are 2 steps to storing files in the repository:

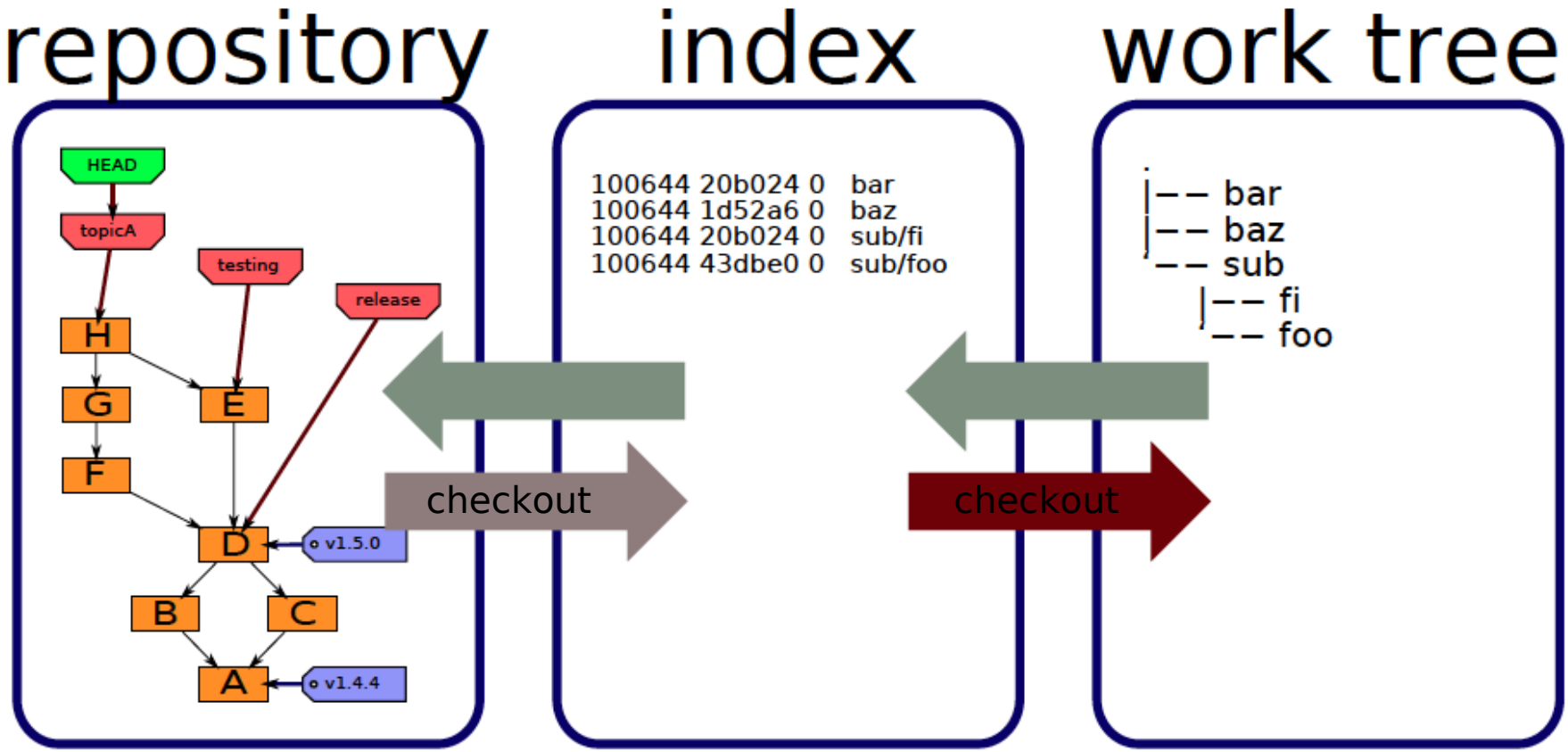
1) mark while files you want to add/update/move/delete
2) actually "commit" the changes

Commit: update work in the repo



"**commit**" updates the repository using the index (staging area). Each "commit" stores a snapshot of some work as a new node (or "version") in the repository.

Checking out a file or version



You can "checkout" individual files or a complete version (snapshot) of the repository using any version. Git will quickly rearrrange the "tracked files" in the working copy.

What to Include in a Repo?

Files you should include are:

1. source code

2. design and notes, such as an architecture notebook, UML diagrams

3. documentation

4. (optional) project plans and project documents

What NOT to save in a repo.

In a Git repository you should not save:

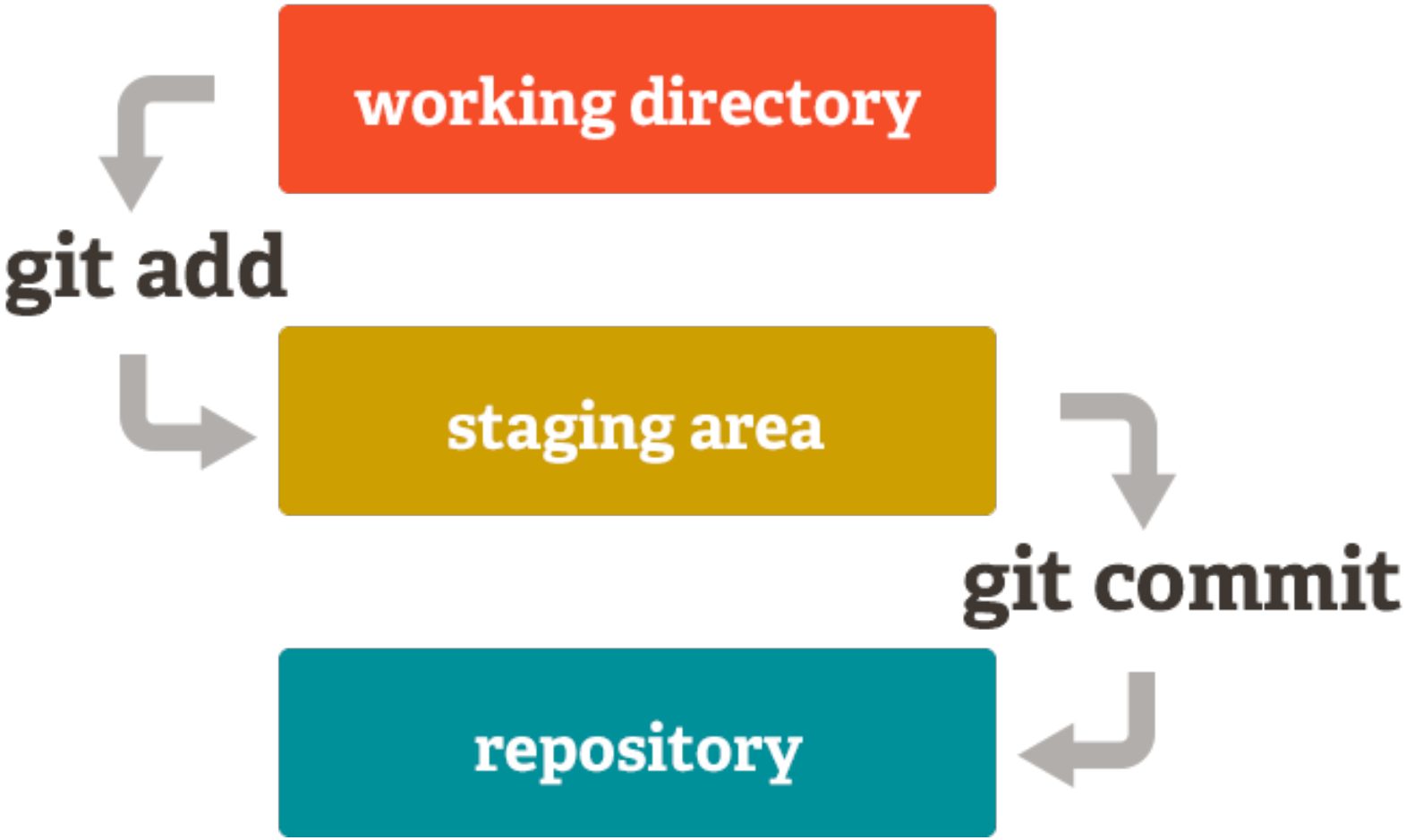
1. compiler output
2. anything you can recreate with a build tool. But OK to save "release" distributions or executable.
3. virtual environments, such as Python virtualenv.

   can be recreated using a `requirements.txt` file
4. IDE or editor project files, such as .vscode/ or .idea/
5. temporary files and log files

Working with Git

The essential commands to know are:

`git init`	create a new repository
`git status`	check status of your working copy
`git add`	add files to staging area
`git commit`	commit staged work to repo
`git log`	view history of commits
`git help` *cmd*	get help on any git command


working directory
git add
staging area
git commit
repository

Create a repo for existing code

Verify that git is installed and on your search PATH:

cmd> **`git --version`**

Change to your project directory

cmd> **`cd workspace/myproject`**

Create a new (empty) repository

cmd> **`git init`**

Adding Files

You need some files to add.

Create a file README.md containing description of your project. You can use Markdown mark-up.

cmd> **`edit README.md`** (any editor you like)

Add the file to "staging area"

cmd> **`git add README.md`**

Check Status & Commit

Always check status before commiting, to avoid mistakes.

cmd> **`git status`**

```
On branch master
Changes to be committed:
   (use "git rm --cached <file>..." to unstage)
new file:   README.md
```

Commit the files, -m "message" is a commit msg.

cmd> **`git commit -m "Initial code checkin"`**

```
 [master (root-commit) 51a8d5a] initial code checkin
 1 file changed, 5 insertions(+)
 create mode 100644 README.md
```

View History

There are several command for this, depending on system:

cmd> **`git log --oneline`**

cmd> **`git log1`**

cmd> **`gitk`** (a graphical tool)

```
51a8d5a (HEAD -> master) initial code checkin
```

Normal Workflow

1. Do some work and save files
2. a) Add files to staging area

   ```
   git add file1 file2 ...
   ```

   b) Rename files

   ```
   git mv oldname newname
   ```

   c) Delete files previously added to repo

   ```
   git rm unwanted_file
   ```

3. Check status: `git status`
4. Commit changes: `git commit -m "helpful msg"`

Working with a Remote Repo

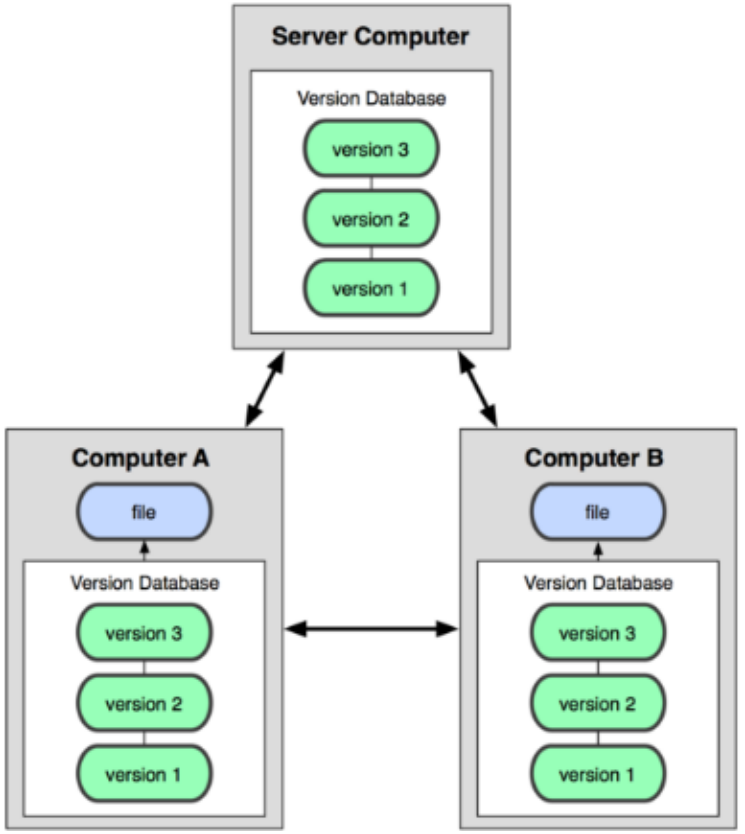
Git is a distributed version control system.

git was invented to manage the Linux kernel source code, with thousands of developers in over a hundred countries.

You can have many repositories on the net, called "remotes".

They may all be different!

There is no "master" repository -- all are equal.



Git Hosting Sites

You can create free git repositories on these sites, for individual or team projects.

Github - https://github.com

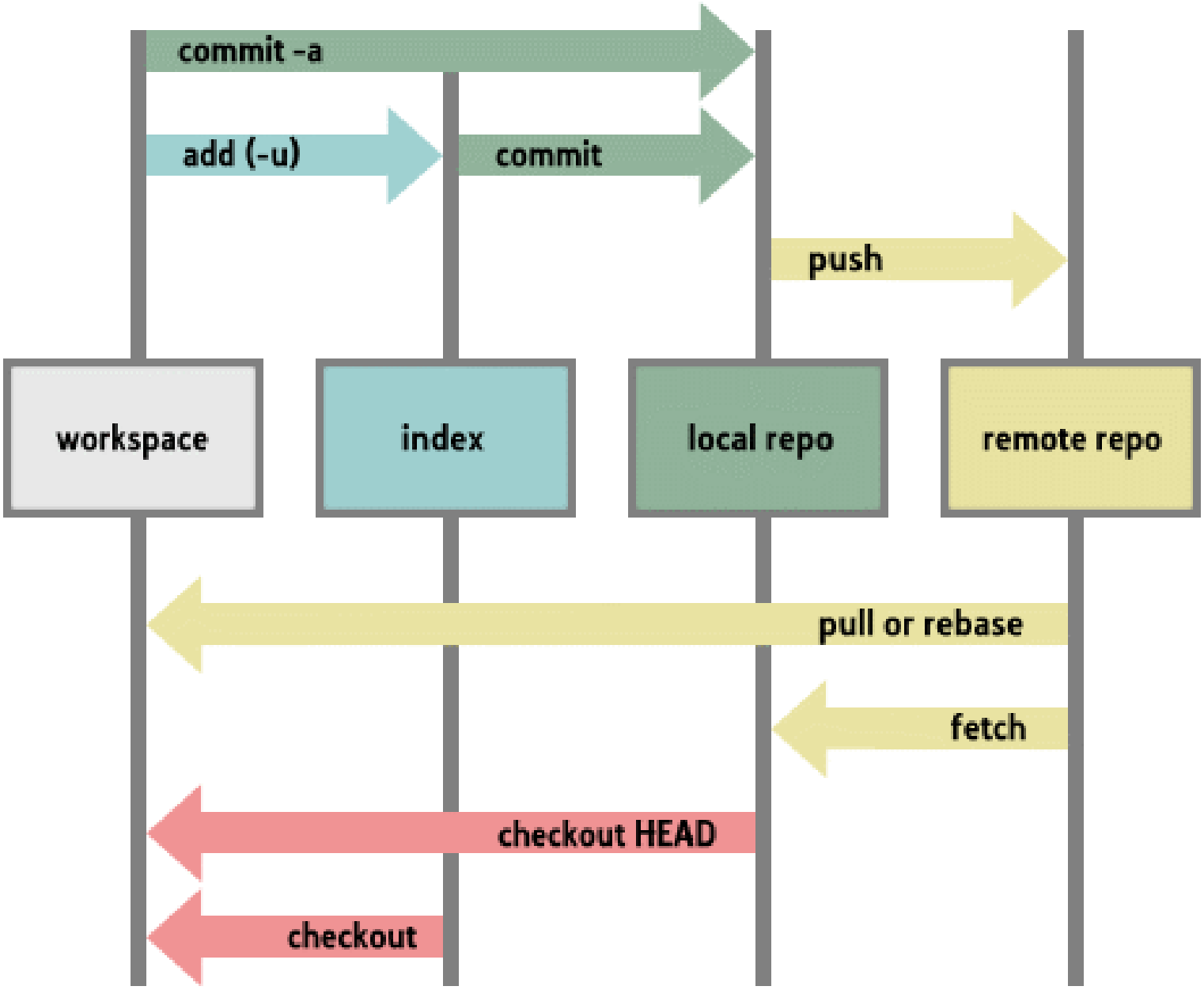
Bitbucket - https://bitbucket.org

GitLab - https://gitlab.com

Commands for Remotes

Common commands for using a remote repo are:

`git clone`	copy remote repo to your computer
`git remote add`	define URL of a remote repository
`git remote -v`	list remotes, with URLS
`git push`	"push" local repo updates to remote
`git pull`	download and merge remote updates
`git fetch`	download remote updates, but don't merge into your working copy


commit -a
add (-u)
commit
push
workspace
index
local repo
remote repo
pull or rebase
fetch
checkout HEAD
checkout

.gitignore

To avoid accidentally adding unwanted files a repository, add a file named `.gitignore` to the top directory of your repo. This is common practice.

`.gitignore` contains names or *patterns* for files and directories that git should ignore -- that is, never add to a repository (unless you force it to).

See next slide for example.

Github can create a `.gitignore` for you when you create a repository! This is a good way to see an example of what you might put in .gitignore.

Examples what not to save

```
__pycache__
*.py[cod]    this means *.pyo or *.pyc or *.pyd
.coverage    output of code coverage app
*.log
# Intellij files
.idea/
*.iml
# Eclipse, Pydev, and VSCode files
.settings
.project/
.vscode/
# Virtual env files (common directory names)
venv/
env/
```

.gitignore for a Python project

Write one filename, directory name, or pattern to match per line.
Lines beginning with # are comments. Blank lines are ignored.

```
__pycache__
*.py[cod]
.coverage
htmlcov/
*.log
# Intellij files
.idea/
*.iml
# Eclipse, Pydev, and VSCode files
.settings
.project
.vscode
# Virtual env files (common directory names)
venv/
env/
```

References

Tutorial from the Git home:

https://git-scm.com/docs/gittutorial

Git Intro (PDF) from U. of Washington

https://courses.cs.washington.edu/courses/cse391/18su/lectures/9/391Lecture09-Git-18su.pdf

Git Tutorial on freecodecamp

https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/