

What is a Version Control System (VCS)?

- Version control (source/revision) is about **managing changes** to documents (source code files)
- Logical way to **organise** and **control** versions
- VCS is an **application** (or part of) to manage the version control
 - Git, SVN, ...
 - Google Drive, Dropbox, Wikis

Why Use Version Control?

- Software exist in multiple versions
 - also deployed
- To maintain, important to access the right version
- Also, parallel development
 - features and/or fixes

Evolution of VCS

- Stage one, **manual** VCS
 - basically keep multiple **copies** of the source code (version)
 - **distribute** source files between developers, e.g., **mail**, **floppies**, etc.
 - **track** everything manually

Evolution of VCS

- Stage two, local VCS
 - SCCS and RCS
 - local file systems, and local locking

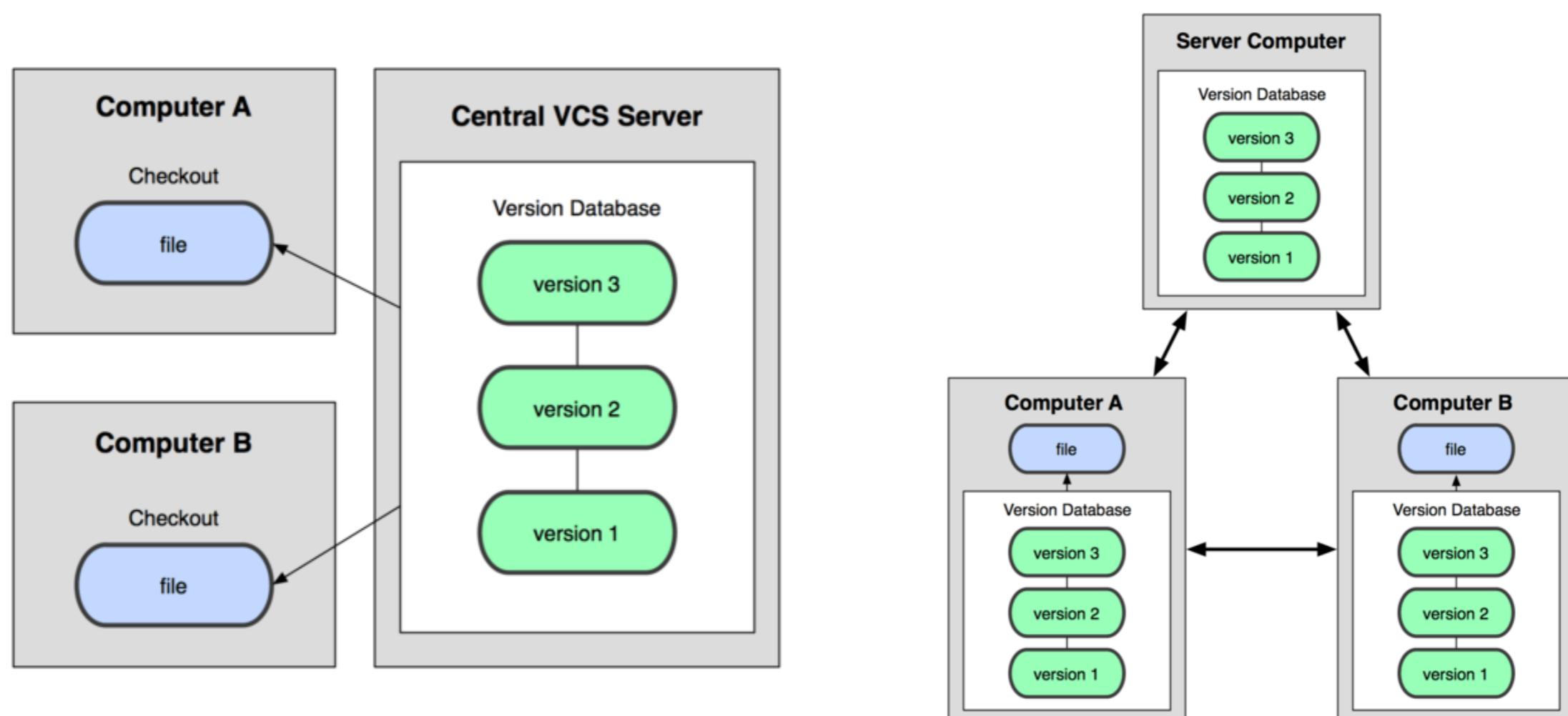
Evolution of VCS

- Stage three, centralised VCS
 - CVS and SVN
 - all operations happen against a central server
 - local copy and remote repository

Evolution of VCS

- Stage four, decentralised VCS
 - Git, Mercurial, Bazaar
 - multiple “copies” of the repository that are kept in sync
 - “everything” is available locally

Evolution of VCS



Important Concepts

- Repository / Working Copy
- Change (List)
- Commit
- Trunk / HEAD
- Conflict / Merge
- History

Git

- Distributed version control
- Developed by Linus for Linux development
- Distributed
- Strong support for non-linear development
- Efficient for large projects

Getting Started

```
git clone <url | path> [dir]
```

- Clones a repository (more or less the entire project history)
- One of two ways to start using Git
- Locally (file system), or remotely (HTTPS, SSH, Git)

Git Basics

- Inspecting
 - status, log, diff
- Modifying
 - add, rm, mv, commit
- Use `git help [<cmd>]` to get help!

Configuring Git

```
git config [--global] <key> <value>
```

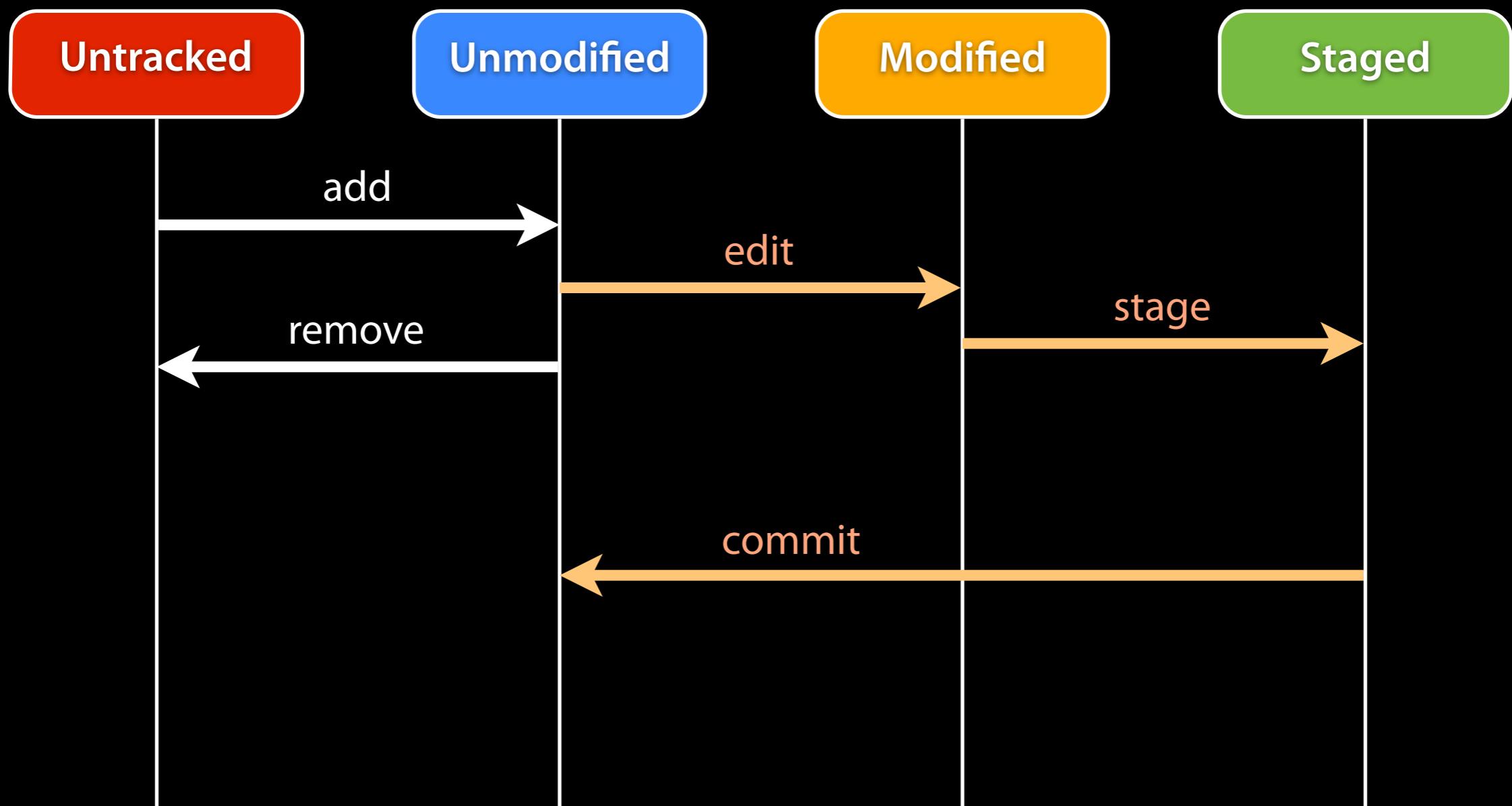
- Allows you to change **settings**, such as **identity**, **editor**, etc.
- Use **--global** to set information **across** all your **repositories**

Demo

File States

- **Untracked:** not managed by Git
- **Unmodified:** tracked, but not changed
- **Modified:** tracked and changed since last commit
- **Staged:** will be saved at next commit

Work Flow / Concepts



Add to the Repo

- Use `git add` to **add** files to the **staging area (index)**
- A **staged** file will be “**saved**” the **next** time you **commit** (basically create a snapshot of your repo)
- You can add **files** or **directories**, several at a time (and as many times as you like)

Committing Your Changes

- Saves a **snapshot** of your **current** repository
- **Staged** files become **tracked** by Git and are now considered “**unmodified**”
- When you commit, Git will ask you to **comment** on your **changes**

Removing and Moving Files

- You can also **remove** files ...
 - `git rm`
 - ... and **move** files
 - `git mv`
- Note that when you remove a file, you will remove the **local copy** (as well)

Logging and Diffing

- You can use `git log` to show the commit history
 - so, good commit messages are important!
- And, you can use `git diff` to see changes between current and commit or local and remote

.gitignore

- The `.gitignore` file allows you to specify **files** that should **never** be tracked by Git
 - such as `.class` files or `.DS_Store`
 - **Text file** that lives in your repository
 - with a list of **patterns** of files and directories to ignore
 - **Github** can help you generate sane **defaults!**

Branches

- A **branch** is a **copy** of the project **state** at a **specific time**
 - a fork
- There is a **main** branch, master (or trunk)
 - and as **many branches** as you create

Why Branch

- Feature development
 - maintain a **stable** version while you work on something **experimental**
- Bug fixes
- Release management
 - the **DAT255** repo has a **VT2013** branch

Branching

- Use `git branch` to **manage** branches
 - `git branch <name>`
- Use `git checkout` to **switch** branches
- Other **commands** work as **expected** with branches

Merging

- When we want to **integrate** or **incorporate** changes across branches, we **merge**
 - `git merge <branch_to_merge>`
- There can be **conflicts**
 - that you might have to **resolve** manually

Playing With Others

- Use `git clone` to get started
- To **update** to or from a **remote** repository
 - `git fetch`, `git push` and `git pull`
 - note, `pull` **fetches** and **merges**
- Use `git remote` to check / manage **remote** repositories

Branches

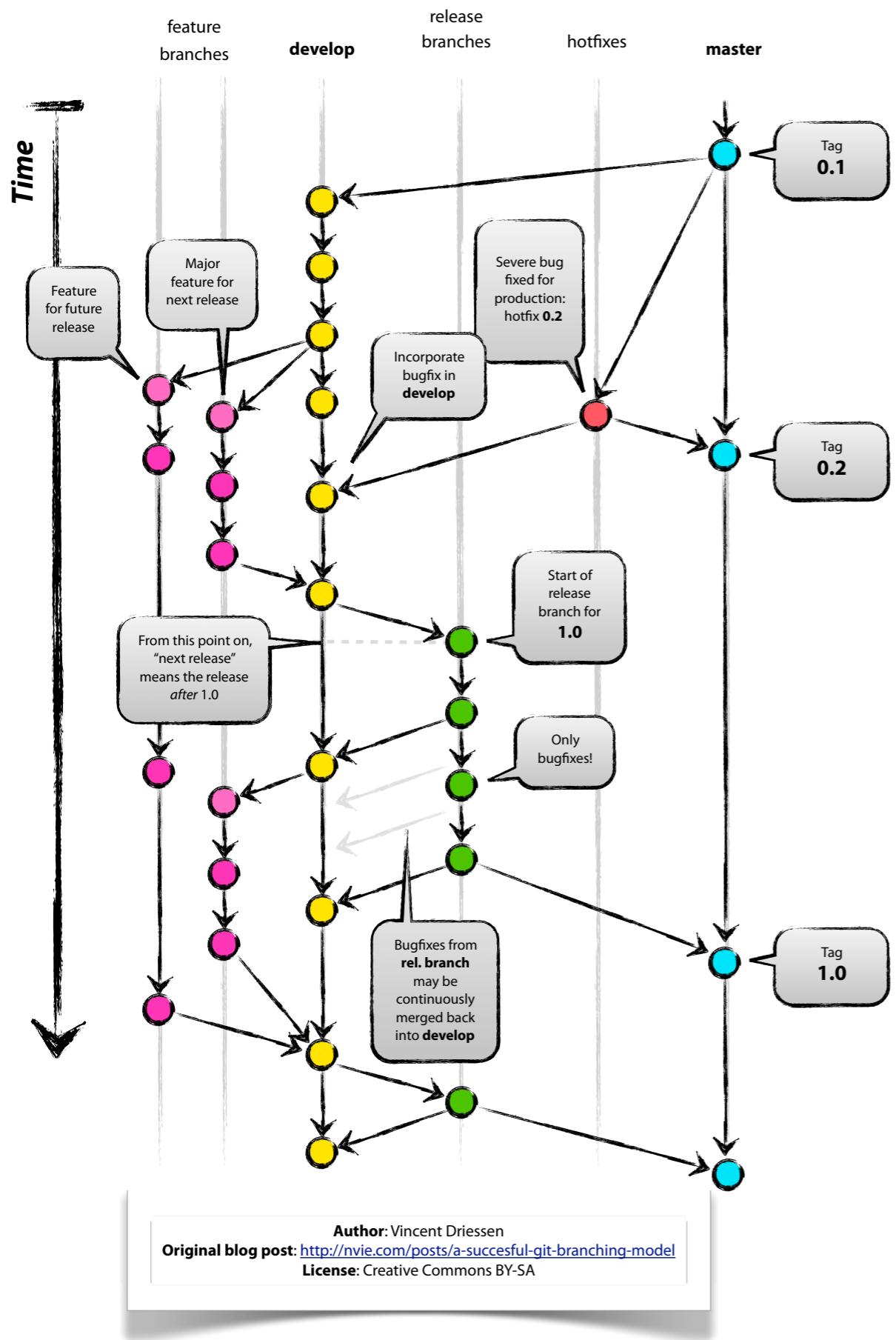
- Branches makes it more complicated, and working with other developers usually means more branches
- Remote branches are local but cannot be modified
 - you need to create a local branch to change / modify it

Releasing (and Tagging)

- A **release** often has a **name** or a **version** attached to it
- Use **tags** to mark specific **commits**
 - light-weight and annotated
 - `git tag [-a] <tagname> <commit>`
- Use **annotated tags** for **releases**
- Tags are **not pushed by default** (use `--tags`)

Best Practice

- The VCS can be a powerful tool or just another obligation (in this course)
- To make it **powerful**
 - **set it up** properly
 - find a good **structure**
 - **embrace** branching



- Do not use *master* for everything!
- Use development, feature, fixes and release branches
- Tag the various releases
- Keep the branches as part of the history/log!

Demo