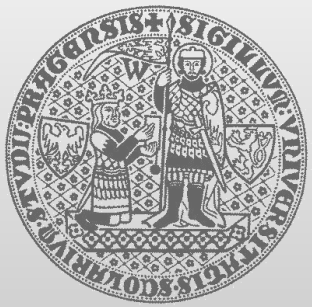
Distributed Version Control

http://d3s.mff.cuni.cz

Pavel Parízek

parizek@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Key concepts

- Each developer uses a private local repository
  - *clone*: full mirror of some existing repository
- Operations performed on the local repository
  - very fast, off-line
- Synchronization
  - Operations *push* and *pull*
  - Exchanging code patches

Department of Distributed and Dependable Systems D3S

Comparing distributed and centralized VCS

- Centralized
  - Everything visible in the central repository
  - Private branches (work) not possible
- Distributed
  - Private repositories (and branches) useful for experimental development

Tools

- Git
- Mercurial
- Bazaar

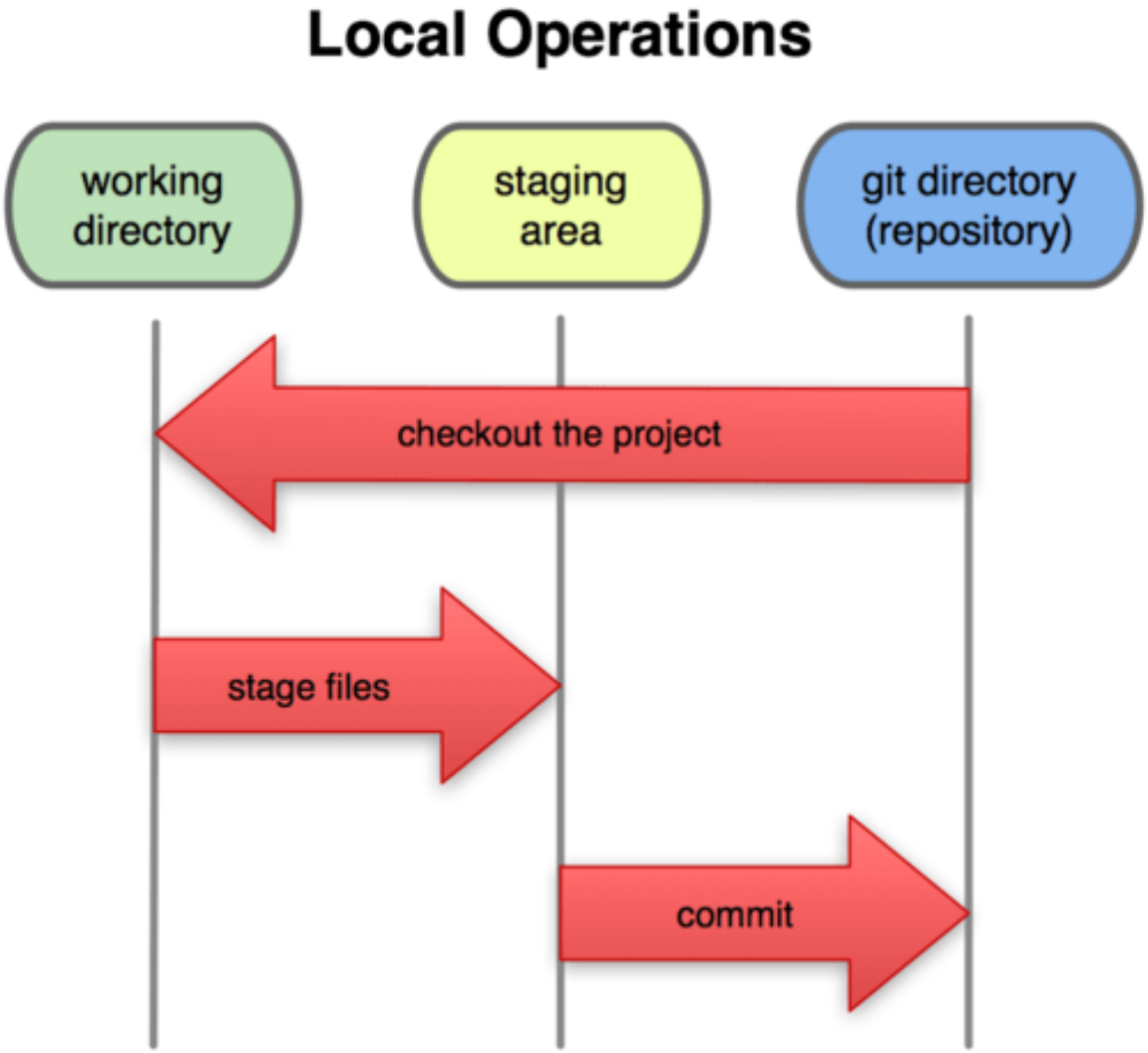
Git

Main features

- Versions: snapshots of the project (working dir)
- Committed revisions form a direct acyclic graph
  - Multiple “latest” versions (leaf nodes)
- Each commit has an author and committer
  - Distributing changesets via patches (email)
- Whole repository stored in `.git` (files, metadata)
- Confusing for most people (good for advanced users)
- Commands have names similar to SVN

Department of Distributed and Dependable Systems D3S

Usage scenario



Picture taken from http://git-scm.com/book/

Task 1

- Configure your identity
  - `git config --global user.name "<your full name>"`
  - `git config --global user.email "<your email address>"`
- Stored in `$HOME/.gitconfig`

Basic commands

- Create repository in the current directory: `git init`
- Print status of the working tree: `git status`
- Start tracking new files: `git add <work dir path>`
- Add files to the staging area: `git add <path>`
- Commit staged modifications: `git commit -m "..."`
- Print uncommitted unstaged changes: `git diff`
- Print staged uncommitted changes:

  ```
  git diff --staged
  ```

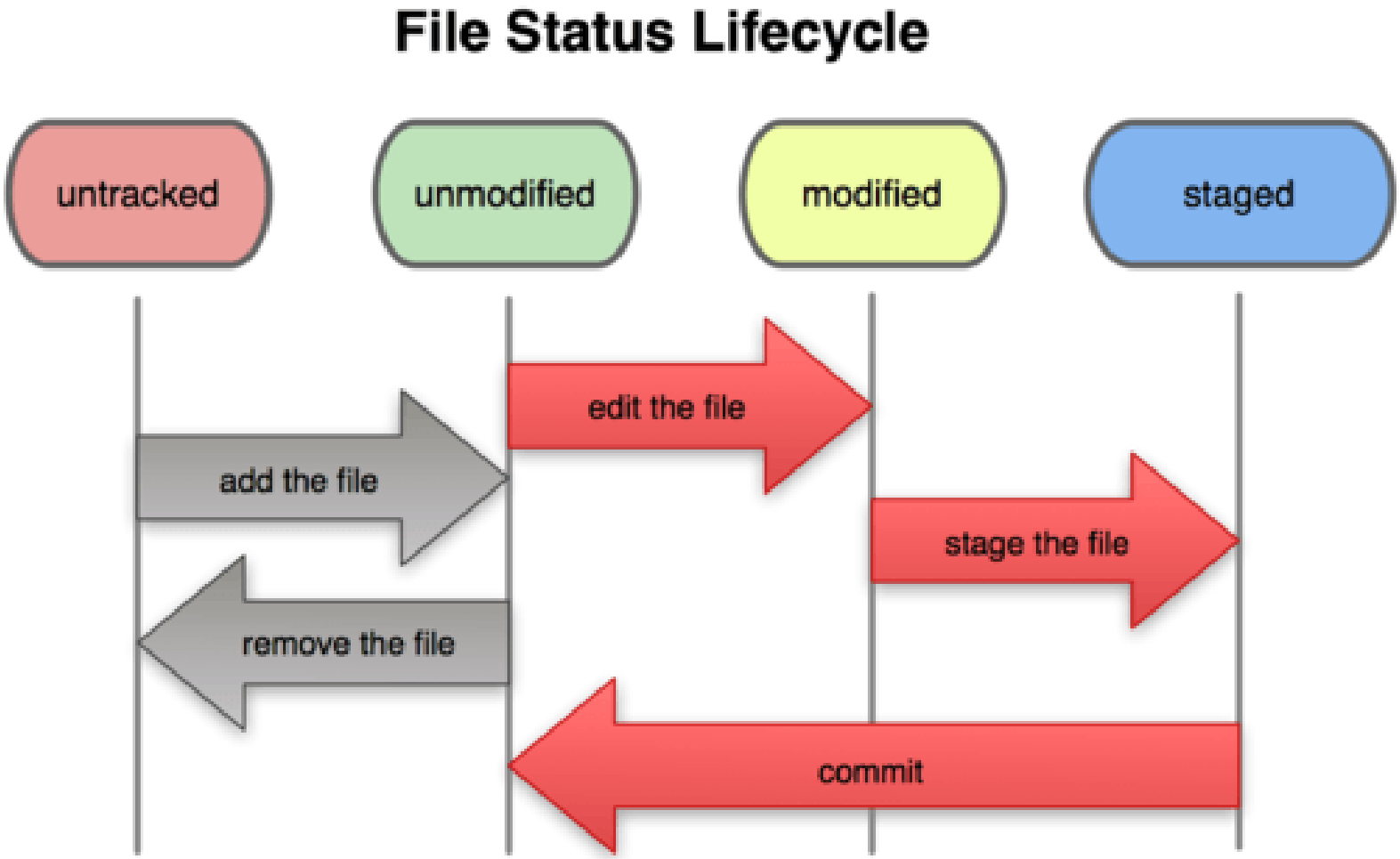
- Automatically stage every tracked file and commit

  ```
  git commit -a -m "..."
  ```

- Revert modifications: `git checkout -- <path>`

Department of Distributed and Dependable Systems

File status lifecycle



Picture taken from http://git-scm.com/book/

Task 2

- Create repository in a specific directory
- Create some new files (e.g., hello world)
- Print current status of your repository and the working directory
- Stage all the new files
- Print current status
- Modify one of the files
- Print current status
  - Inspect differences from the previous invocation
- Commit all staged modifications
- Print current status

Managing files

- Make the given file untracked

```
git rm <work dir path>
```

- Renaming file (directory)

```
git mv <old path> <new path>
```

Pick your changes

- Full interactive mode: `git add -i`
- Select patch hunks: `git add -p`

Project history

- List all the commits

```
git log [-p] [-<N>] [--stat]
```

- More options

```
[--pretty=oneline|short|full|fuller]
[--graph]
[--since=YYYY-MM-DD]
[--until=YYYY-MM-DD]
[--author=<name>]
```

Task 3

- Try out file management commands (`rm`, `mv`)
- Play with the "`git log`" command
  - Explore different parameters (`-p`, `-<N>`, `--stat`, `--pretty`, `--graph`)
- Run the program "gitk" and try it
- Make some changes to a particular file and use interactive staging

Department of Distributed and Dependable Systems

Using remote repositories

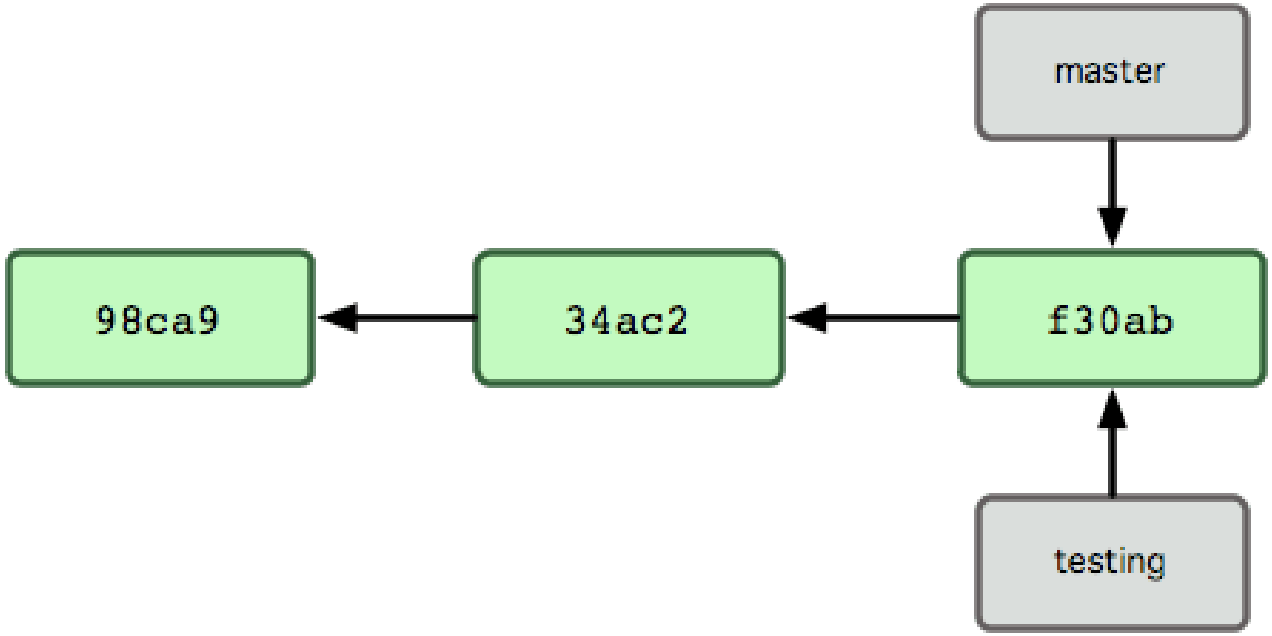
- Clone a remote repository in the current local directory: `git clone <repo url>`
- Get recent changes in all branches from the remote repository: `git fetch origin`
- Get recent changes in the "master" branch and merge into your working copy: `git pull`
  - Announcements via *pull requests*
- Publish local changes in the remote repository: `git push origin master`

Department of Distributed and Dependable Systems D3S

Branches in Git

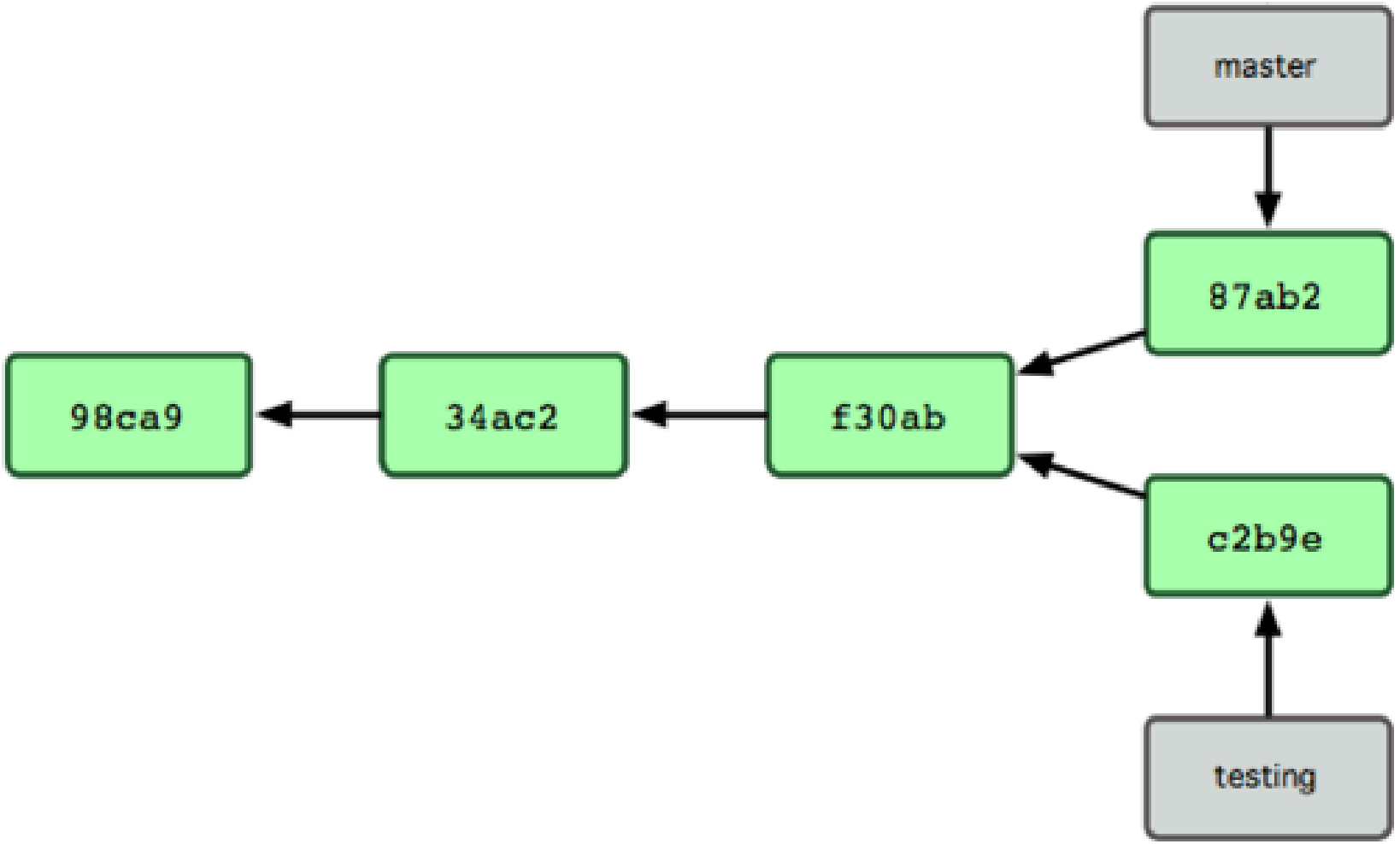
Branches in Git

- Branch: pointer to a node in the revision DAG
- Default branch: **`master`**
- Commit: branch pointer moves forward



Picture taken from http://git-scm.com/book/

What happens after concurrent modification



Picture taken from http://git-scm.com/book/

Branches in Git: commands

- Create new branch: `git branch <name>`
- Switch to given branch: `git checkout <name>`
- Shortcut: `git checkout -b <name>`
- Merge branch into current working directory

  ```
  git merge <branch name>
  ```

- Deleting unnecessary branch

  ```
  git branch -d <branch name>
  ```

- List all branches: `git branch [-a]`
  - Current branch marked with `*`

Comparing branches

- `git diff <branch 1>..<branch 2>`
  - Compare heads of the two branches
  - Note the characters '. .'

- `git diff <branch 1>...<branch 2>`
  - Print changes on the branch 2 (e.g., master) since the branch 1 (feature) was created from it
  - Note the characters '. . .'

Three-way merge

- Common ancestor
- Target branch
- Source branch

- Conflicts happen also with Git
  - Standard markers `<<<<<<<` `=======` `>>>>>>>`
  - Marking resolved files: `git add`

- Graphical merging tool: `git mergetool`

Task 4

- Create new branch B and switch to it
- Modify some files and commit them
- Switch back to the master branch
- Modify some files and then commit
- Merge your branch B into the master
- Delete the now unnecessary branch

- Try switching branches with uncommitted changes in the working copy
- Try graphical merging tool on some conflicts

More advanced features

- Symbolic names of versions
  - HEAD, HEAD~1, HEAD^2
- Using stack of unfinished changes (stashing)
- `git reset`
  - Several variants: clear the index, undo some commits
- `git rebase`
  - Replaying changes done in a branch onto another branch
  - Very powerful command but also tricky
- Modifying committed history
  - e.g., commit messages
- Ignoring certain files
  - List patterns in the file `.gitignore`
- Tagging: `git tag`
- Bare repository
  - No working copy

Mercurial

- Basic principles: like Git
- Simpler learning curve
- Commands very similar
  - `init`, `clone`, `add`, `commit`, `merge`, `push`, `pull`
- Better support for Windows

Department of Distributed and Dependable Systems

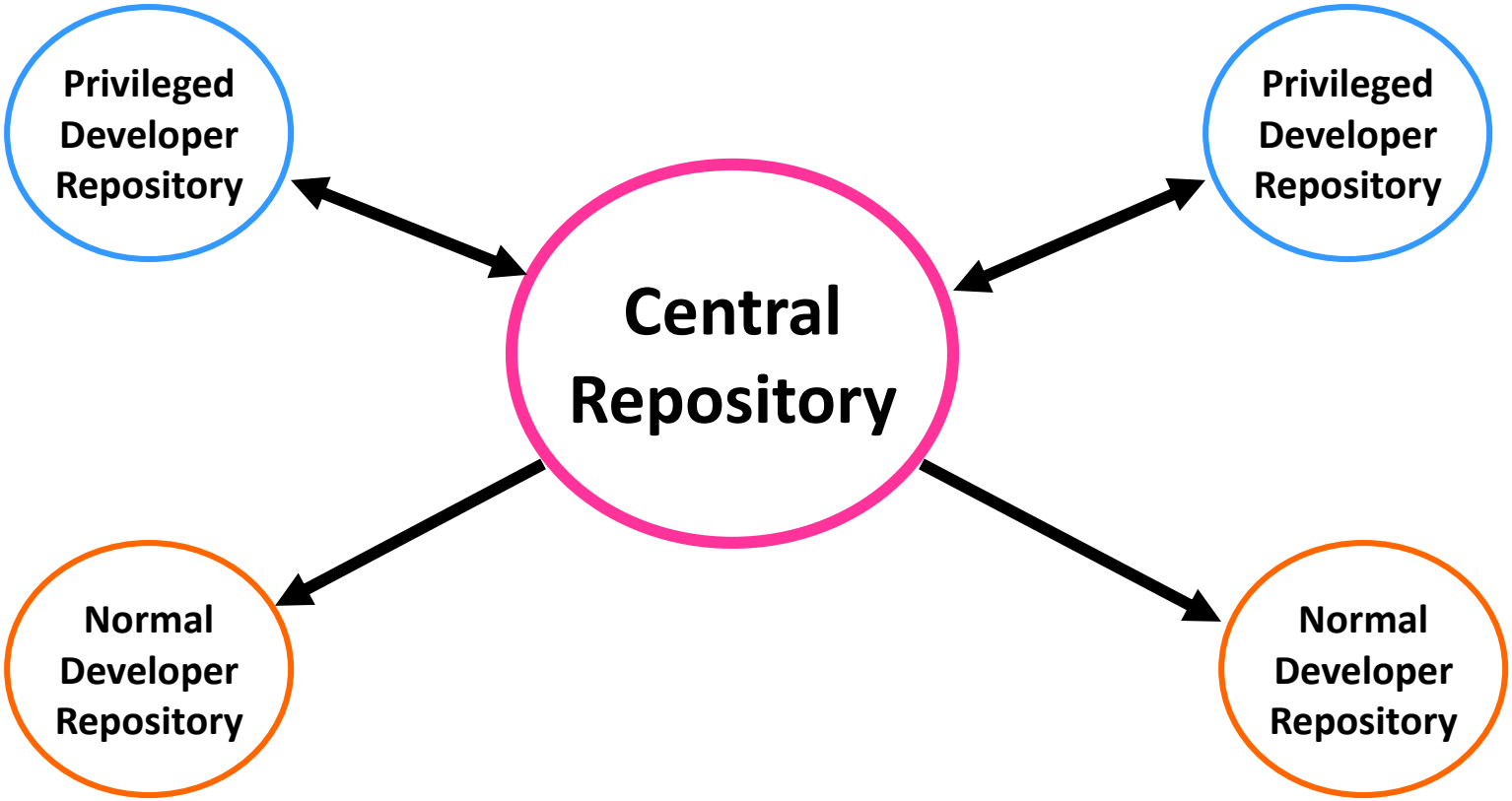
Work-flow models (cooperation)

Work-flow models (cooperation)

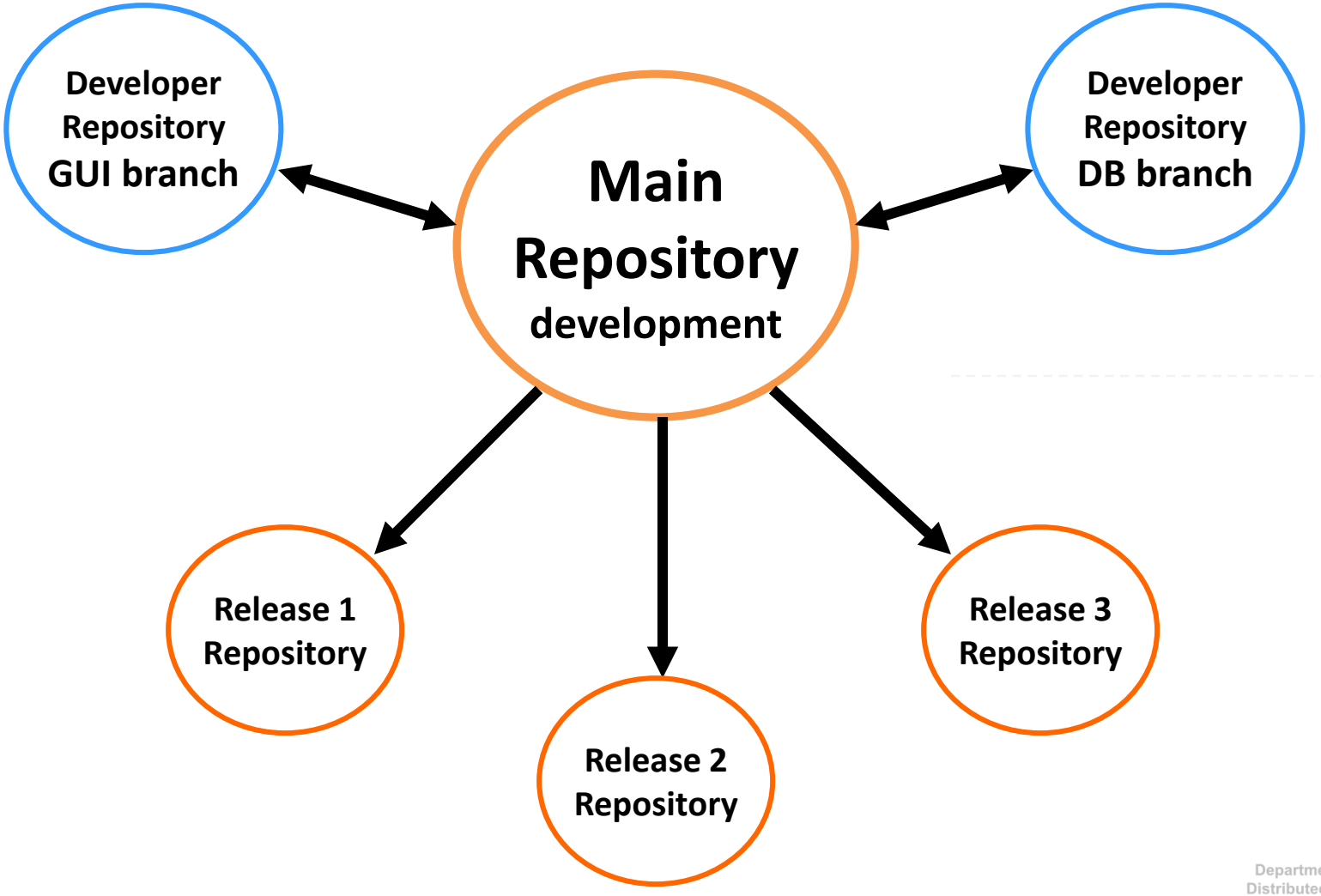
- Anything possible technically with DVCS
- “Network of trust” between developers

- Examples
  - Single “central” repository
  - Multiple release repositories
  - Many public repositories
  - Total anarchy

Single “central” repository

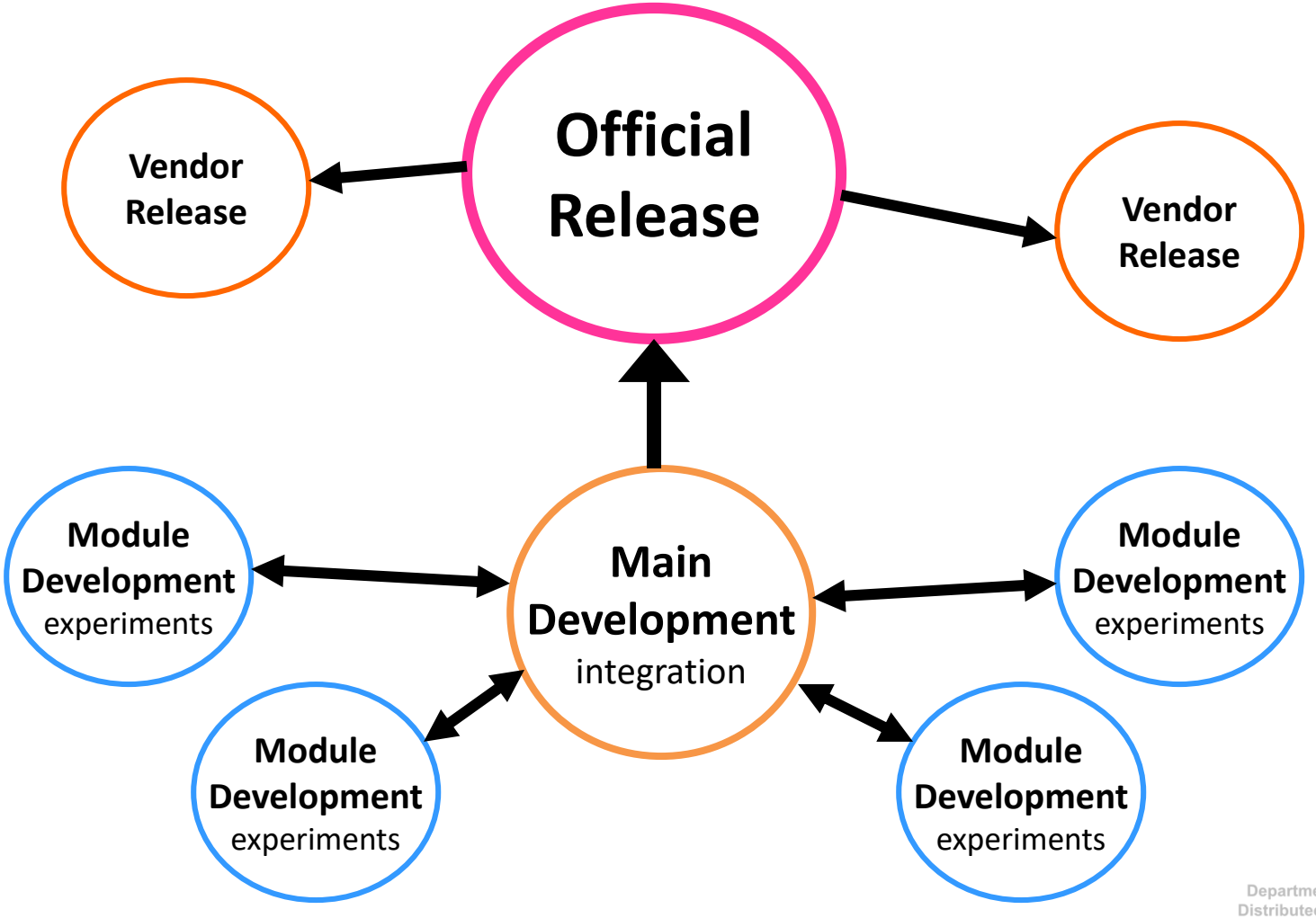


Multiple release repositories

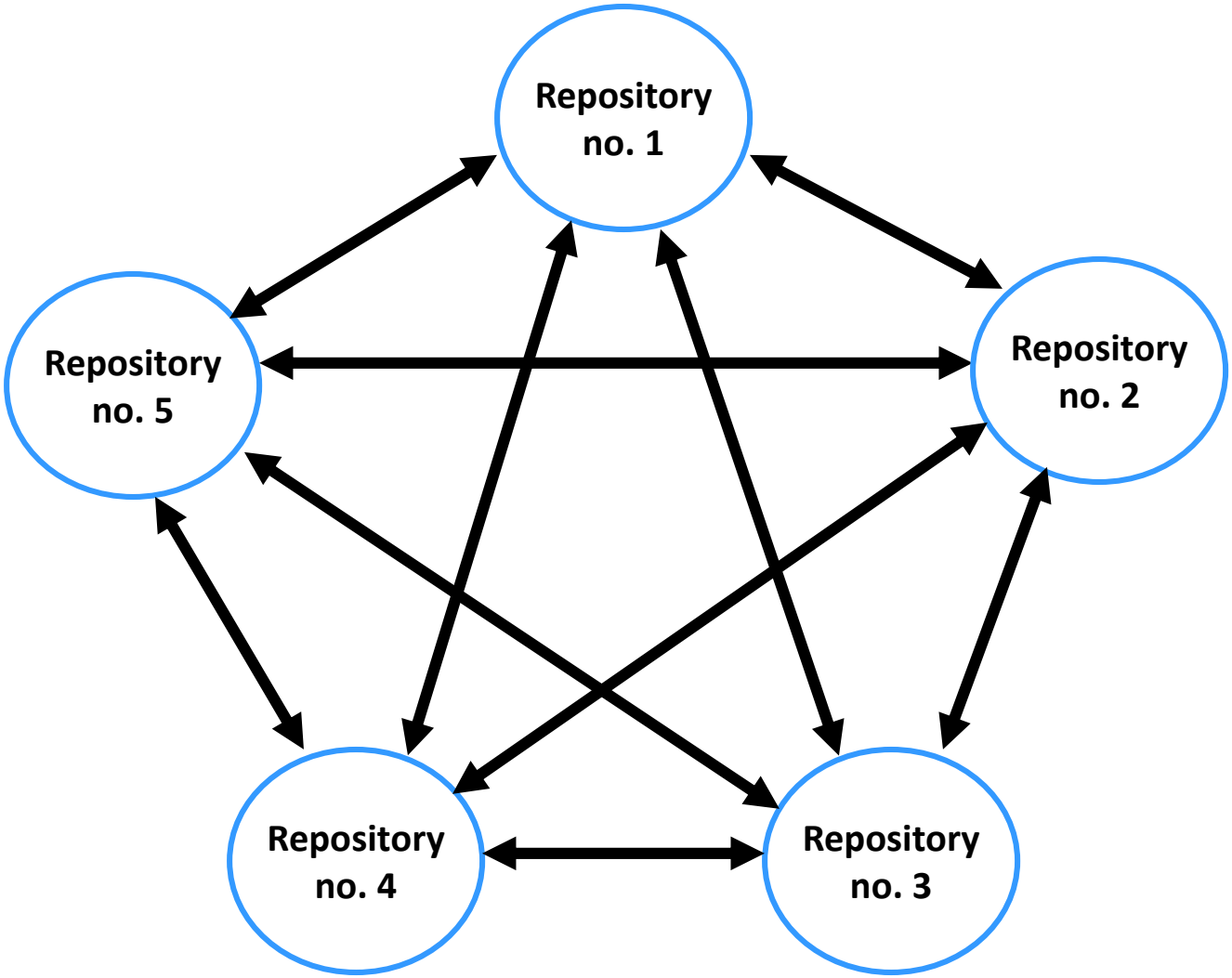


Many public repositories

- Linux kernel



Total anarchy



Links

- Git documentation
  - http://git-scm.com/doc
- Mercurial
  - http://www.mercurial-scm.org/, http://hgbook.red-bean.com/
- Repository servers
  - https://github.com/
  - https://bitbucket.org/
  - https://gitlab.com/
- Tools
  - Git for Windows (http://msysgit.github.io/), TortoiseGit (Win), SmartGit (http://www.syntevo.com/smartgit/)
  - TortoiseHg (Mercurial GUI, Windows)
  - SourceTree (https://www.sourcetreeapp.com/, Git and Mercurial)

Homework

- Assignment
  - http://d3s.mff.cuni.cz/~parizek/teaching/sdt/
- Deadline
  - 22.10.2018 / 23.10.2018

Department of Distributed and Dependable Systems