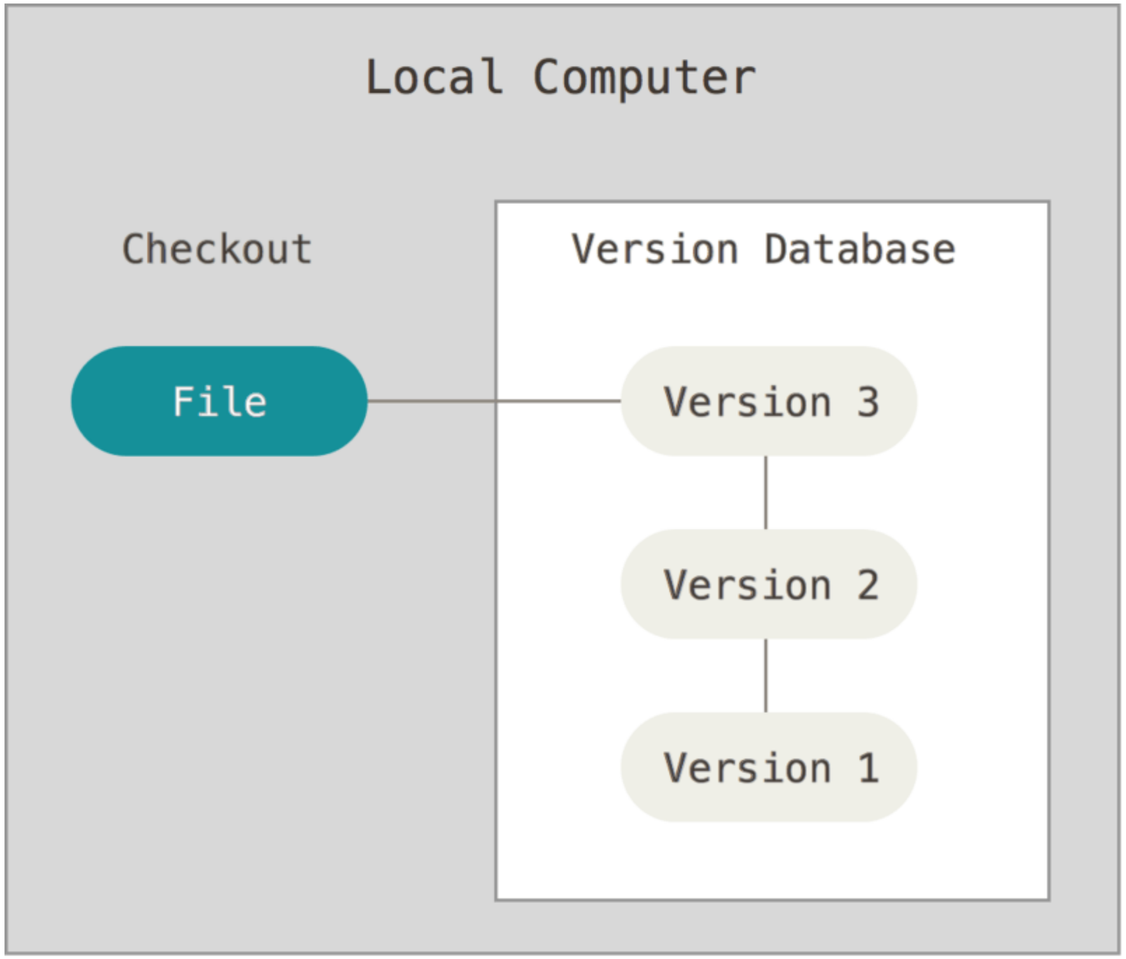
Practical Git

Chun Li

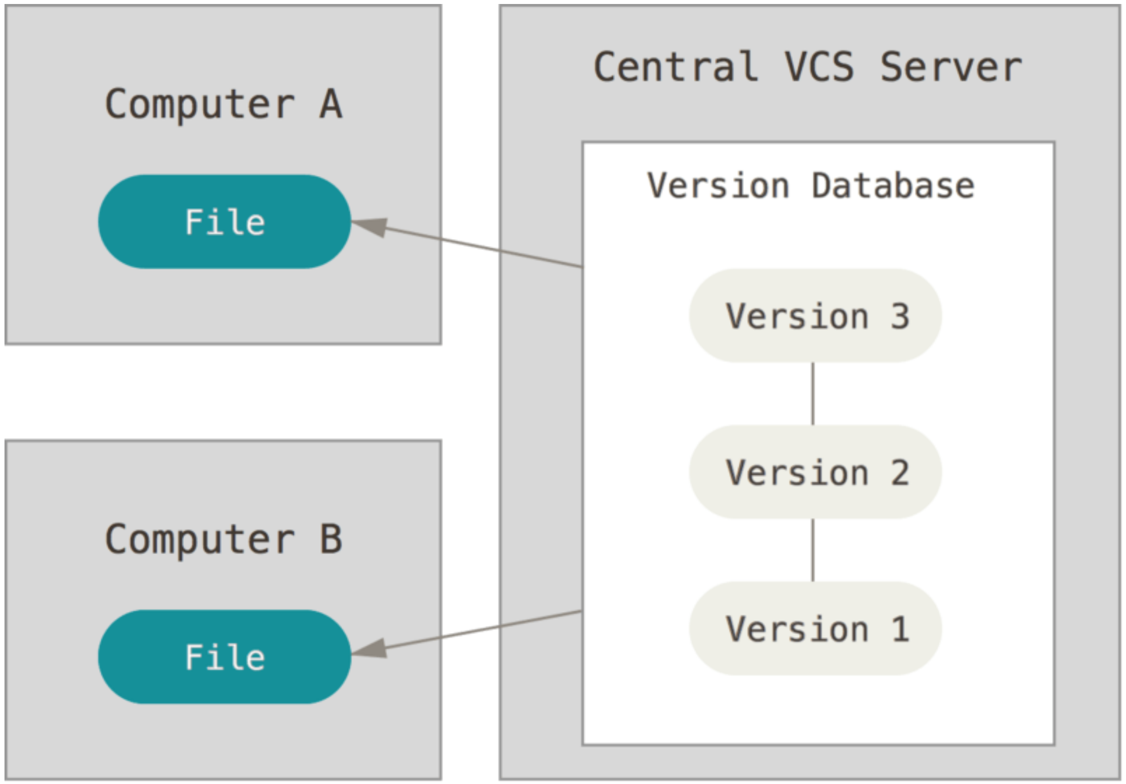
Intro

- Git is a distributed version control system
    - Named after it's creator, Linus Torvalds

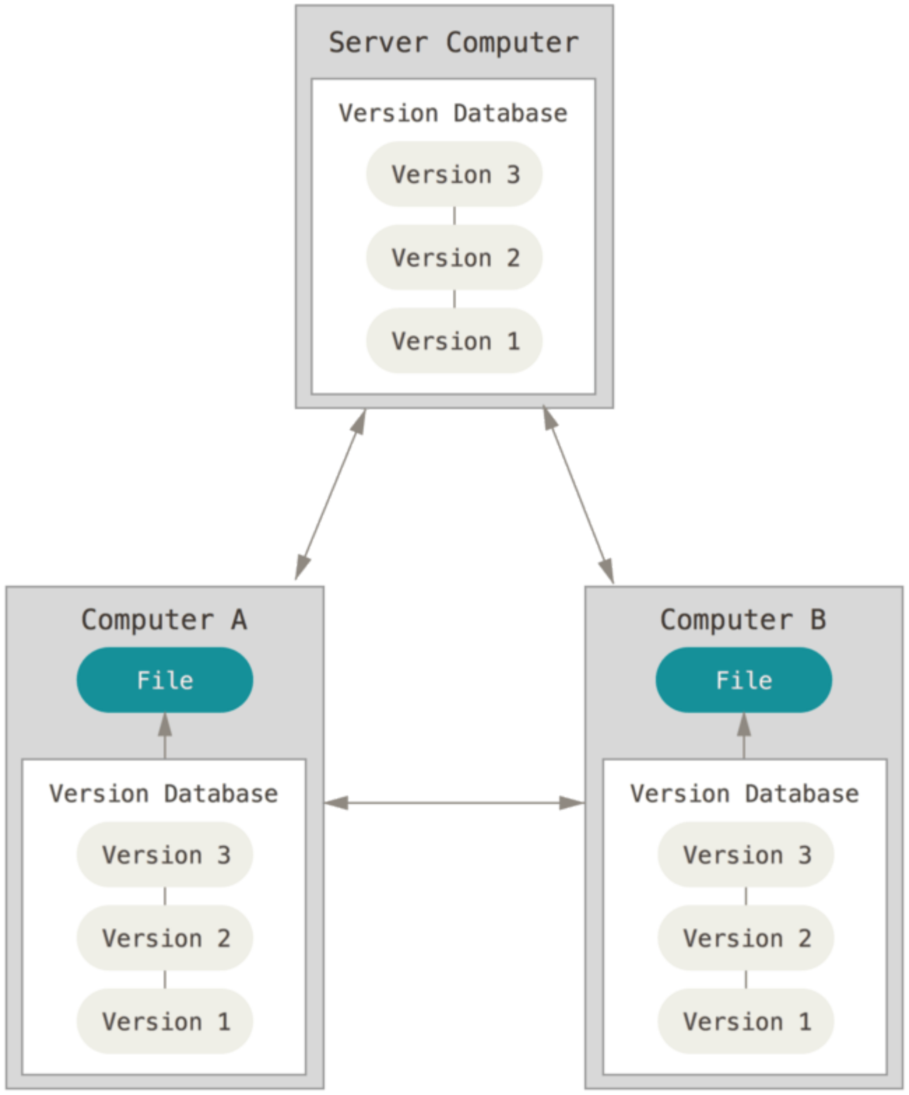
Local VCS


Local Computer
Checkout
File
Version Database
Version 3
Version 2
Version 1

Centralized VCS

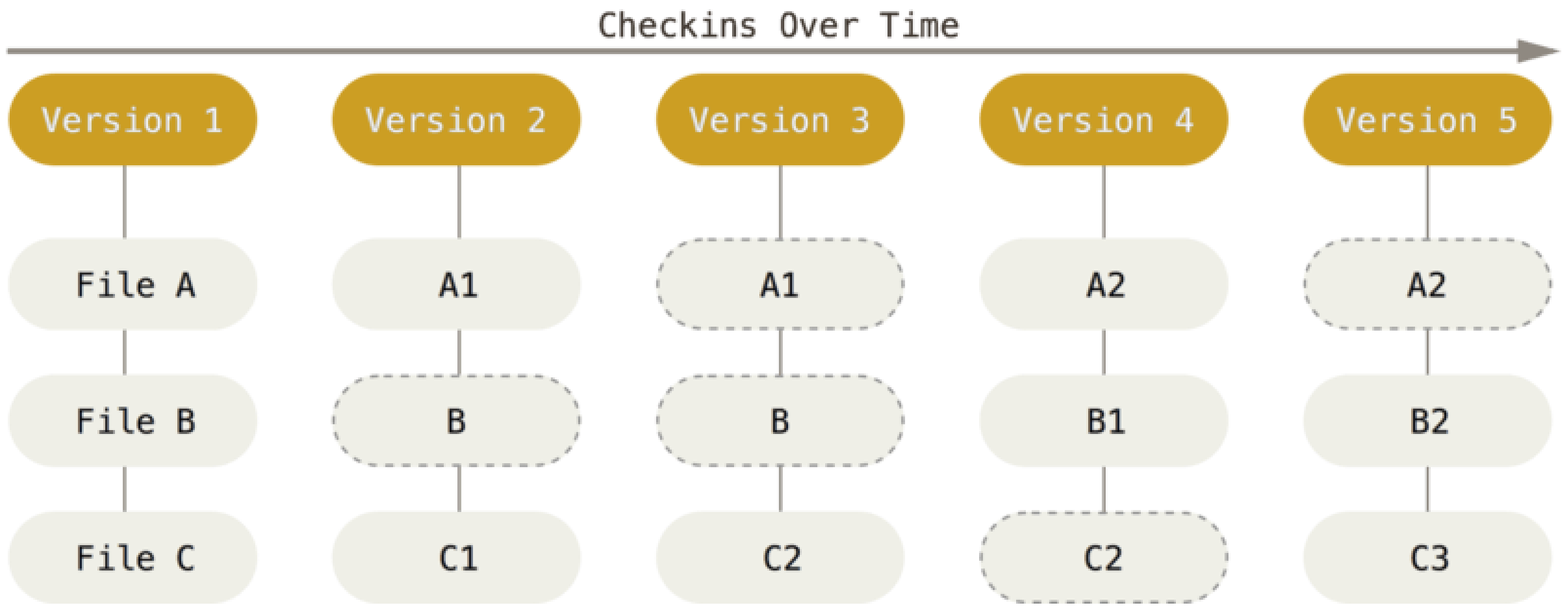


Distributed VCS



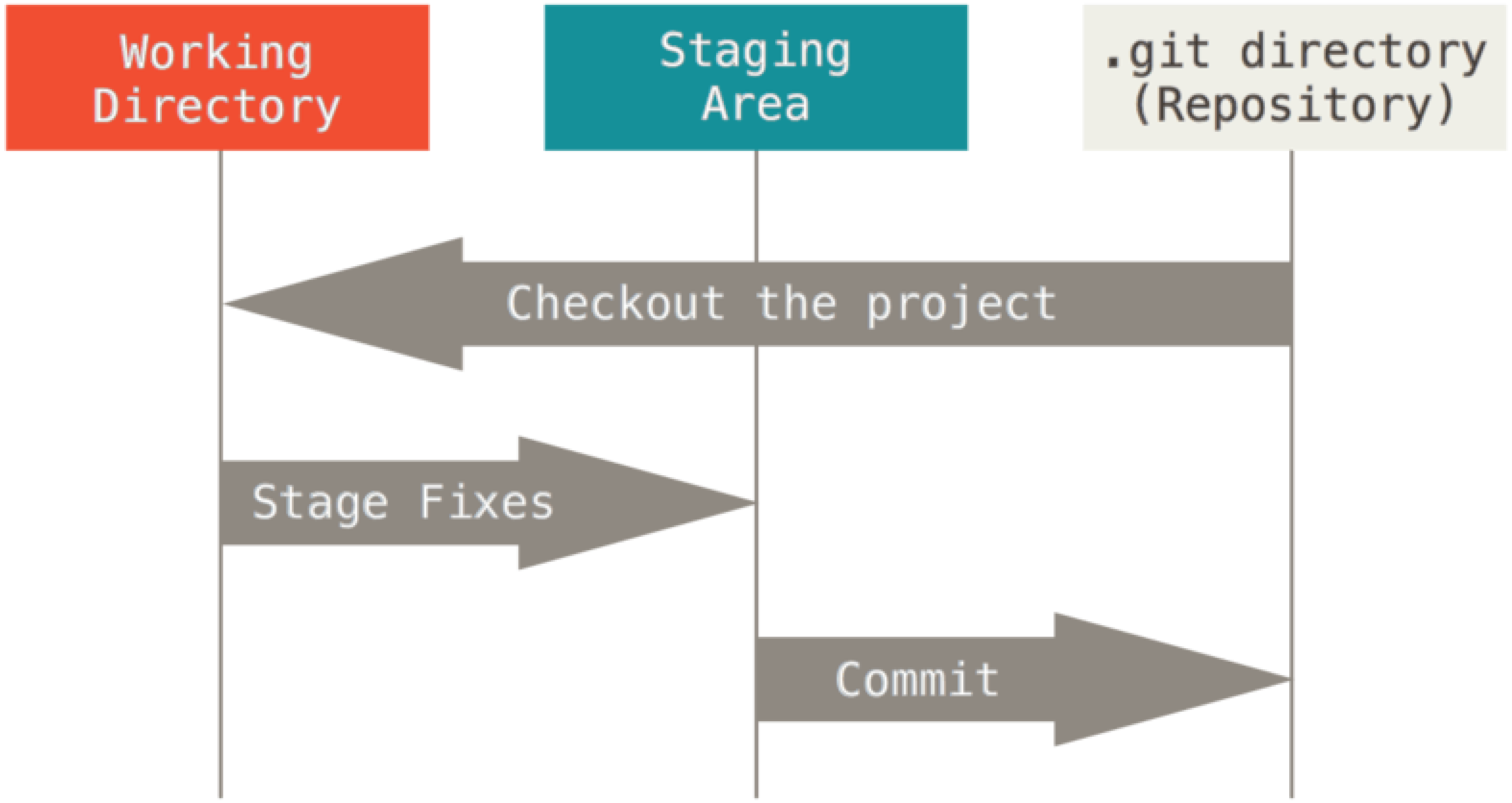
Snapshots

- Git stores it's data as snapshots of the file, rather than a diff
    - If a file has not changed it just stores a reference to save space
    - Your git history is just a tree of snapshots



Three states

- Files generally exist in one of these three states
- One last nice thing about git: most operations *add* information, so it's rather difficult to permanently lose information once commited



Let's get started

`man git-<command>` gives help page for a git command

`git init` or `git clone <repo>` to start a new repo

Config

`git config <config.key>` to print config value
`git config --global <config.key> <value>` to set config value

`--system` ➡️ `/etc/gitconfig`
`--global` ➡️ `$HOME/.gitconfig` ⭐ usually you want this one
`--local` ➡️ `.git/config` this is the default

❗ Don't forget to set `user.name` and `user.email` before you start commiting!

Staging

`git status` prints out what files are modified and staged

`git add <file>` starts tracking a new file, or stage modified changes
`git rm <file>` stages removal of file
`git mv <file>` stages file rename

`git reset HEAD <file>` to unstage a file
ℹ️ you can use any `tree-ish` in place of `HEAD` to reset a file to any point in time, like `HEAD^`, `HEAD~10`, or a commit `SHA-1` hash.

`git checkout -- <file>` discards current unstaged changes
❗ cannot be reverted!

Staging

If you ever forget, run `git status`; git will remind you what to do 🙂

```
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committ
  (use "git checkout -- <file>..." to discard changes in

        modified:   README.md
```

Staging

`git diff` shows unstaged changes in your working directory

`git diff --staged` shows staged changes

Commiting

`git commit [-m <message>]` commits staged files.
`git commit -a` stages modified files that are being tracked before commiting.

`git reset --soft HEAD^` to undo last commit and put changes in staging area
ℹ️ `--mixed` resets the index (also unstages the files)
❗ `--hard` resets the index and the working tree (i.e. discards the last commit)

Commiting

`git log` lets you browse past commits

- `-<num>` for the last `num` commits
- `--author=<author>` filter by author
- `-S<string>` search by string
- `--after=<date> --before=<date>` filter by timerange
  - Can be a string like `2006-07-03 17:18:43 +0200`
  - Or relative like `2.weeks`

`--pretty=<fmtstring>` for formatting

- `oneline` is a useful fmtstring

`--graph` to print graph

`git lola` is a popular alias

`git show <commitish>` lets you see the full commit

Tags

`git tag <tag>` creates a lightweight tag
`git tag -a <tag> -m <msg>` creates an annotate tag

- usually recommended to use annotated

`git push <remote> <tagname>` or `git push --tags` to push a tag

- git doesn't push tags by default

Remotes

`git remote -v` to list all remotes

`git remote add <remote> <url>` to add new remote

`git remote rename <remote> <newremote>` to rename a remote

`git remote set-url <remote <newurl>` sets remote url

`git remote remove <remote>` deletes a remote

Branching

`git checkout <revision>` swaps out working directory
`git branch` lists all branches
`git branch -d <branch>` deletes a merged branch
`git branch --no-merged` prints all unmerged branches
`git checkout --track origin/<branch>` to start tracking remote branch
`git push -u origin <branch>` to push to new remote ref
`git branch -u <remote_ref>` to start tracking ref

`git clone` actually sets up the `origin` remote, `master` branch, and `origin/master` remote ref, which is why push and pull work out of the box

`git pull` is actually a `git fetch` on remote refs, and then `git merge`

Branching - Merge

`git merge <branch>` merges `<branch>` into current branch

- creates an empty *merge commit* with 2 parents

On merge conflict: fix changes and use `git add`. Use `git commit` to finalize merge.

`git reset --hard HEAD` to abort merge

Branching - Rebase

`git rebase <branch>` rebases your current branch onto `<branch>`

`git rebase --onto <onto_branch> <base_branch>` rebases your current branch onto `<onto_branch>` using `<base_branc>` as the base

`git rebase --interactive` gives you full control over how you want to rewrite history

On merge conflict: fix changes and use `git add`. Use `git rebase --continue` to continue

`git rebase --abort` to abort merge

Extras

`git cherry-pick <commitish>` to apply a particular commit

- use `-x` to create a standardized commit message
- `git` also has a "patch hash", so it will properly merge later on (although you might have duplicate commits in the history)

`git stash`, `git stash list` and `git stash pop` for quickly storing diffs

`git reflog` to list past objects `HEAD` was pointed at

- `git checkout <hash>` can recover "lost" data up to a month ago

Extras; Old School

`diff` and `patch -p0` are still useful

`git format-patch <since>` for emailing patches since `<since>`, and `git apply <patch_file>` to apply or `git am <patch_file>` to apply and commit.

Useful Aliases

```
st = status
ci = commit
br = branch -v
co = checkout
df = diff
ds = diff --staged
lola = log --graph --decorate --pretty=oneline
           --abbrev-commit --all --date=local
ls = ls-files
unstage = reset HEAD
out = "!git log origin/master..$(git rev-parse
      --abbrev-ref HEAD)"
in = "!git log $(git rev-parse --abbrev-ref HEAD)
     ..origin/master"

# Show files ignored by git:
ign = ls-files -o -i --exclude-standard
```

Recommended Resources for Help

- `man git-<command>`
- *Pro Git*, Chacon and Straub - Chapters 1, 2, 3
- Google/Stack Overflow it
- http://justinhileman.info/article/git-pretty/git-pretty.pdf
- Poke the person next to you

References

Chacon, S., & Straub, B. (2014). *Pro Git*. Berkeley, CA: Apress.

Git Reference: https://git-scm.com/docs

Stack Overflow