
git

Why Git?

It’s fast

It’s distributed

It has cheap branching

It’s well-supported

It’s better than SVN
(for our use case)

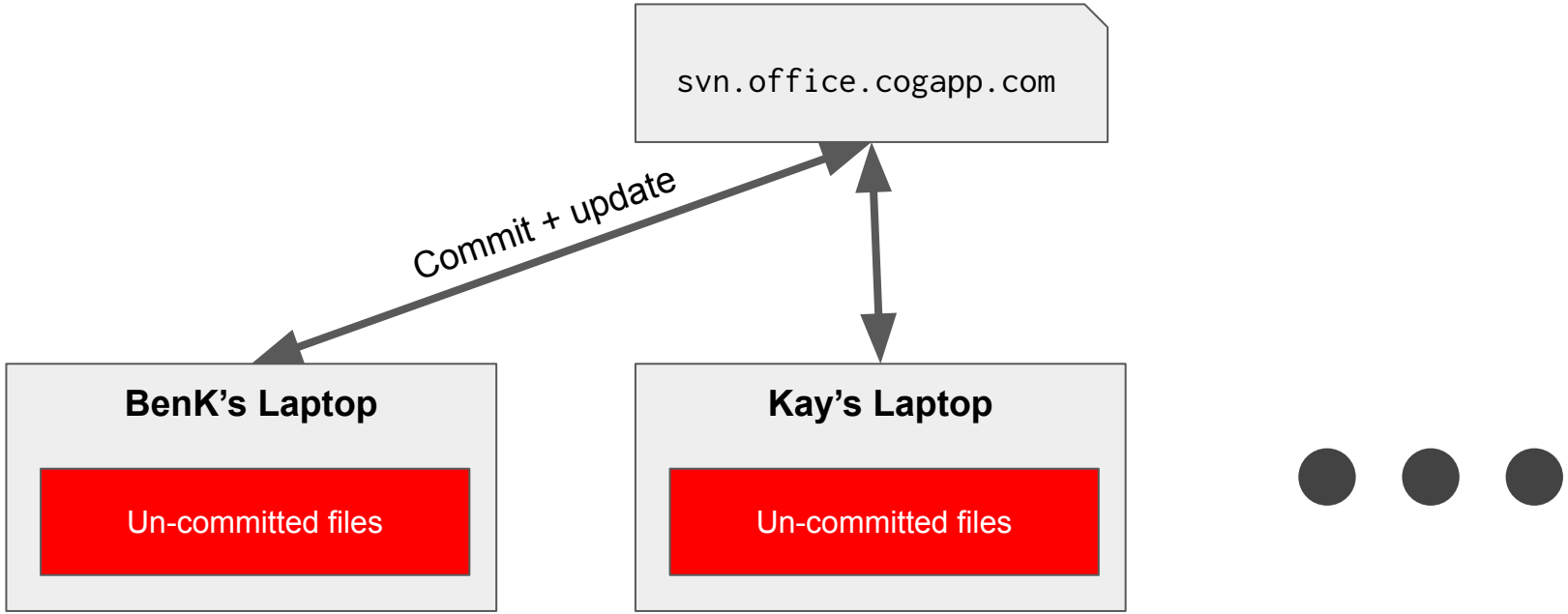
The downsides?

Some uncommon operations can get very complicated

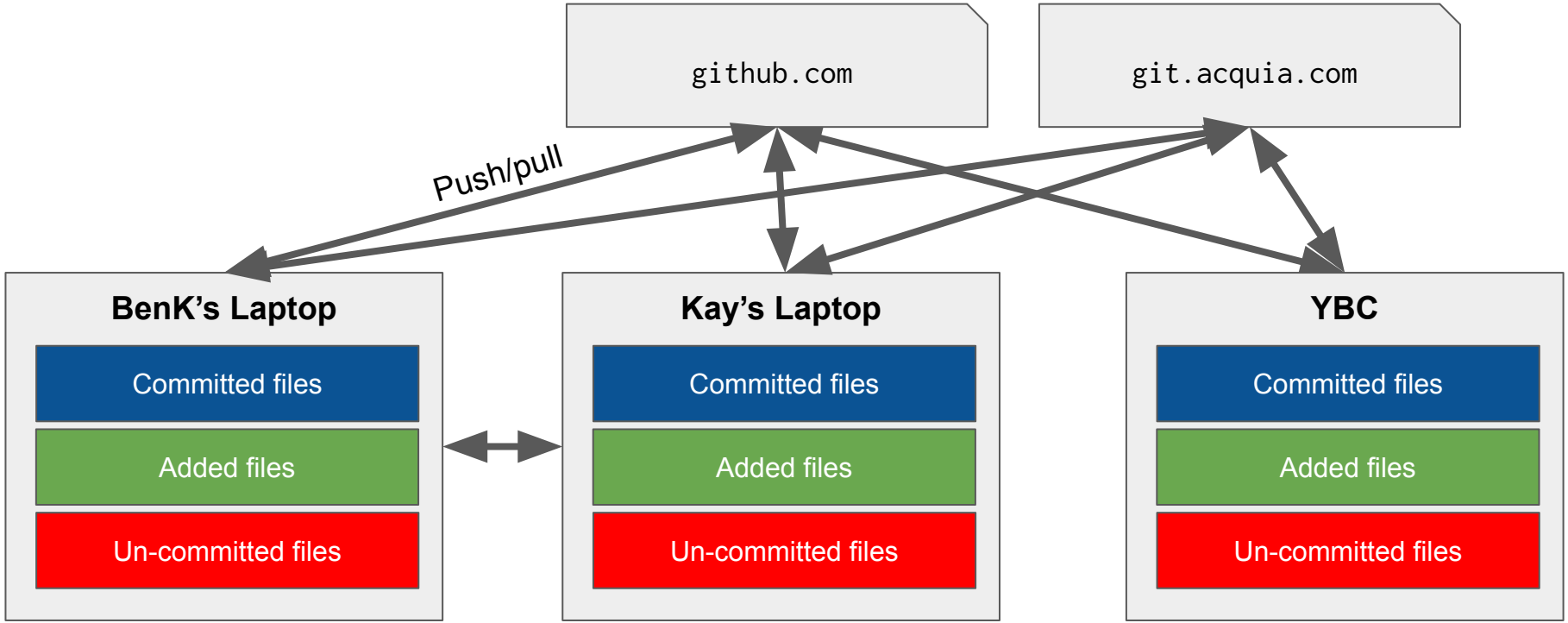
It's powerful enough to let you really shoot yourself in the foot

The interface can be a bit obtuse (e.g. reset vs. revert, checkout vs. branch)

The SVN model



The Git model



Basic Git commands

```
git clone
git status
git checkout
git add
git commit
git push
git pull
git fetch
git merge
```

A sample feature workflow

```
> git clone git://...
> git checkout -b my-feature-branch
> git push -u origin my-feature-branch
... Write code
> git add .
> git commit
... Repeat until complete
> git push
```

A sample reviewer workflow

```
> git checkout my-feature-branch
... Review
> git checkout develop
> git pull
> git merge my-feature-branch
> git push
```

A note on cloning

Always prefer the git:// protocol to https://

You can use this with your SSH config, and you don't need to enter passwords

Just put this in your `~/.ssh/config` file (update your key location as needed)

```
Host github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/github/github
```

Some useful commands

```
git log --oneline

git log branch1..branch2 --oneline --reverse

git log origin/mybranch..mybranch --oneline --reverse

git clean -d

git stash (pop)

git checkout -- <file>

git reset <sha> (--hard/--soft)
```

Some other useful commands

```
git cherrypick

git bisect

git reflog
```

Set your editor

When you do anything that requires an editor (writing commit messages, using rebase) you can tell Git which editor it should use with `core.editor`

The command **must** force the editor to wait for you to save before returning

```
git config --global core.editor vi

git config --global core.editor nano

git config --global core.editor “subl -n -w”

git config --global core.editor “bbedit --wait --resume”
```

Set up autocomplete

There's an official autocompletion script for branches and commands

You can see how to download and set it up here

Set your mergetool

When you need to do a merge from the CLI, Git will use your configured tool

```
git config --global merge.tool <tool>
```

Then when you call `git mergetool`, it will open each conflicted file sequentially

Git natively supports a bunch of options for this (as long as the tool is installed)

I've used Diffmerge, which now seems to have been superseded by Meld

You asked...

Has anyone used Git for creating something that was not a piece of software?

Yes - someone on GitHub is using it to write a novel

But there is a much higher barrier to entry than something like ‘track changes’

It’s also not going to be very helpful if you’re versioning a Word document

If you’re non-technical or your artifact isn’t in plain-text there’s probably something better

Why is Git so bad at tracking moved files?

This is a conscious design choice

Git tracks content and not files

You can get extended history for a file with

```
git log --follow <file>
```

Or for a commit

```
git show -M <sha>
```

Best way to add large files to a repository?

Git LFS seems to be the recommended way.

GitHub also lets you attach files to releases when releasing binaries.

If you're working with file formats Git can compress, don't commit an archive.

What kind of large files do we need to store?

How to handle permissions changes when working with Drupal (Features)?

Change Apache/Nginx to run as your user and not `www-data`, `_www`, etc.

There are other permissions issues with `sites/default` too

Unfortunately these aren't fixed in core, but there are patches available

7.x https://www.drupal.org/project/drupal/issues/1232572
8.x https://www.drupal.org/project/drupal/issues/1925822
8.x https://www.drupal.org/project/drupal/issues/2869916

Should I use global or local config

There are use-cases for both
(but most of the time you'll use global)

Global config is useful for having defaults, and for system-specific settings

Local config is useful for project-specific configuration

e.g. whitespace handling, username and email

Best practice for Git ignoring junk files specific to you (e.g. IDE dotfiles)

Project .gitignore for project-specific files:

- *.pyc
- *.log
- Test artifacts

Global .gitignore for your files (e.g. IDE or system files):

- .DS_Store
- .idea

We should create a standard one for everyone to use

Best practice for Git ignoring junk files specific to you (e.g. IDE dotfiles)

GitHub has a guide on this

(It also has suggestions for language-specific files)

```
touch ~/.gitignore_global
```

```
git config --global core.excludesfile ~/.gitignore_global
```

Should Cogapp have a standard way to write commit messages and decorate pull requests?

Linus Torvalds says: A good commit message looks like this:

```
Header line: explaining the commit in one line

Body of commit message is a few lines of text, explaining things
in more detail, possibly giving some background about the issue
being fixed, etc etc.

The body of the commit message can be several paragraphs, and
please do proper word-wrap and keep columns shorter than about
74 characters or so. That way "git log" will show things
nicely even when it's indented.
```

Where that header line really should be meaningful, and really should be just one line. That header line is what is shown by tools like gitk and shortlog, and should summarize the change in one readable line of text, independently of the longer explanation.

2FA or not 2FA

Why not 2FA?

Should I even squash commits?

Locally: sometimes
(more on this later)

Remotely: in my opinion, no

I think the merge-squash only works very well for small, atomic changes

It works less well when you're adding entire modules as part of the branch

It's also has the potential to add unnecessary complexity to our workflow

How do I use Git effectively through my IDE? What are some good Git tools/extensions/GUI?

I think you'll always run up against issues with this

We found the same problems with SVN

Simplifying the interface always ends up hiding useful information

Probably fine for the basics, but for complex operations use the CLI

I find PHPStorm has decent tools: not sure about other GUIs

How do I stop line endings from going doolally?

With core.autocrlf

Set it to enforce newlines with:

```
git config --global core.autocrlf input
```

A note on whitespace

You can set core.whitespace to enforce particular kinds of whitespace

The default settings are pretty permissive

You can make it more opinionated with:

```
git config --global core.whitespace blank-at-eol,blank-at-eof,tab-in-indent,-space-before-tab
```

Or for tabs:

```
git config --global core.whitespace blank-at-eol,blank-at-eof,indent-with-non-tab,space-before-tab
```

You can even set one global rule, and override locally per-project

How can I make my diffs better?

By changing the algorithm

```
git config --global diff.algorithm histogram
```

You can also apply these options manually when merging/rebasing

```
git merge <branch> -s recursive -Xdiff-algorithm=histogram -Xpatience
```

This won't apply to anything merged/diffed by GitHub

How do I work effectively on shared branches?

Run your linting

Run your tests

Automate this where possible

Work locally then rebase before pushing

Accept there will be some overlap

Be excellent to one-another

Rebasing

What are some methods for managing the use of rebase whilst still pushing your branch to a remote?

How can I stop creating “Merged my_branch into my_branch” style commits?

Why should I never ever see the message: “Merge branch ‘develop’ of github.comblahblah into develop”

Why, knowing what `git rebase` is and how it works, does it cause shitfuckery so often (asking for someone else, also they told me they won’t accept not understanding how it works as an answer)

On a side note, if rebase is so great, please also elaborate on why merge isn’t. I’m taking the reins a bit here but maybe it could go something like “rebase is cleaner, but it has tremendous potential to screw things up. Merge is safe and magic and does what you want it to. However it creates a merge commit message, and what could be worse that that?”

The prime directive of rebasing

Don’t rebase pushed commits

Pulling with `--rebase`

This is always a safe operation

You only rewrite your local history

Any merge conflicts you encounter will still happen with a regular pull

You don't get the "merged my_branch in to my_branch" commits

```
git pull --rebase
```

What are some methods for managing the use of rebase whilst still pushing your branch to a remote?

Refer back to the prime directive

Useful for tidying your local commits before pushing

```
git rebase -i
```

If you're the only person using the branch, you can push then rebase later
(This involves force-pushing and rewriting public history)

How to avoid commits like:

```
h87sf0 Fixed tests

oi8rw7 Actually fixed the tests

pasd85 Tweaked config to fix the test

ja65bm Okay, now it is actually working.
```

It depends

If they're all local, use a local rebase to squash them all before pushing

If they're remote and you're the sole user of the branch, you could rebase

If they're remote and other people are using the branch, you're SOL

Are there ways of preventing this in your workflow?
(e.g. running tests locally, trying iffy commits on a test branch?)

What is the most dangerous Git command?

```
git push --force
```

```
git reset HEAD --hard
```

A sample `~/.gitconfig`

```
[user]
  name = Jane Doe
  email = foo@example.com

[core]
  editor = vi
  autocrlf = input
  whitespace = blank-at-eol,blank-at-eof,tab-in-indent,-space-before-tab

[diff]
  algorithm = histogram
```

Questions?