Git

Git (/gɪt/) is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

- Push.Default Strategies
- Per Repo hooks

Push.Default Strategies

I was accidentally updating other branches when pushing my feature branch, and didn't realise why.

Ever pushed to the wrong branch by mistake? what happens when you `git push`? Is your expectation that your local branch is pushed to the remote? This might not be the case:

What happens depends on your `push.default` configuration setting! List push default strategy that is in use `$ git config push.default` for your project and `$ git config --global push.default` for the fallback. To set the value, append it to the command.

Push.Default Strategies

There are 5 options, my understanding is that:

1. `simple` pushes the current branch to the **remote branch with the same name, but doesn't create the branch** if it doesn't already exist. This is the default if you've recently updated git.
2. `current` works like simple but **creates the remote branch**. Handy unless you're accidentally pushing to the wrong remote.
3. `upstream` works like current, but pushes to the **tracking branch which might not have the same name**. To illustrate this, you can set the tracking branch to a different destination, for example: `git branch --set-upstream-to=origin/my-experiment CI-3911-husky`

There are two more strategies to be aware of:

4. `nothing` **does not push** anything. I don't know when you'd want to use this.
5. `matching` pushes **ALL branches that exist remotely**!
   Realising that you're pushing *dev/stage/any other branch* with changes when pushing your feature branch, and having to reset them, is not a fun day at the office. This gets more complex when using multiple remotes! I would avoid this strategy.

Tracking branch vs remote branch

The distinction is that all remote branches are potentially tracking branches, but the tracking branch for your local branch does not have to be on a remote.

You can track local master for example for your feature work.

Note that you can find out your tracking branch through `git status`, in general it should contain the remote to avoid unintended consequences:

```
$ git status
On branch CI-3911-husky
Your branch is up to date with 'origin/CI-3911-husky'.
```

Tips

Finally a tip: please specify a remote after `git push` in your `npm / composer` scripts, so that if the remote is set incorrectly for reasons your team is pushing to the right place.

Finally, Finally 111 latest: if you're not sure about any git command, append `--dry-run` to test it non-destructively. And if you get stuck, I found the Flight Rules helpful.

Corrections

Feel free to comment with any corrections or knowledge, everyone is learning :D

Git going!

Per Repo hooks

This is a little helper script that enables version controlled hooks, and per project hooks:

Examples are:

```
.hooks/pre-push          # version controlled repo hook
.hooks/pre-push.local    # overrides the previous hook

.hooks.local/pre-push    # will be run after the hook above
```

Files and folder ending in `.local` should be git-ignored.

Setup

Add the following helper script to your path:

```
#!/usr/bin/env bash
# name: repo-hooks.sh

#{{{ Bash settings
# abort on nonzero exitstatus
set -o errexit
# abort on unbound variable
set -o nounset
# don't hide errors within pipes
set -o pipefail
#}}}
#{{{ Variables
IFS=$'\t\n'   # Split on newlines and tabs (but not on spaces)
SCRIPT_NAME=$(basename "${0}")
SCRIPT_DIR=$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)
readonly SCRIPT_NAME SCRIPT_DIR
#}}}
```

```
if [ $# -eq 1 ]; then
    hook=$1
else
    echo "Error: please provide exactly one hook"
    exit 1
fi

if test -f ".hooks/$hook"; then
    if test -f ".hooks/$hook.local"; then
        echo "Source +hl hooks/$hook.local"
        source ".hooks/$hook.local"
    else
        echo "Source +h hooks/$hook.local"
        source ".hooks/$hook"
    fi
fi
if test -f ".hooks.local/$hook"; then
    echo "Source +l hooks/$hook.local"
    source ".hooks.local/$hook"
fi
```

Add the following line to the hooks you want to enable this for in `~/.config/git/hooks/`:

```
repo-hooks.sh $(basename $0)
```

Note: Hooks run under the current user, so don't run this on untrusted repositories that might contain evil hooks!