

your code with others.

COMMIT OFTEN

Committing often keeps your commits small and, again, helps you commit only related changes. Moreover, it allows you to share your code more frequently with others. That way it's easier for everyone to integrate changes regularly and avoid having merge conflicts. Having few large commits and sharing them rarely, in contrast, makes it hard to solve conflicts.

DON'T COMMIT HALF-DONE WORK

You should only commit code when it's completed. This doesn't mean you have to complete a whole, large feature before committing. Quite the contrary: split the feature's implementation into logical chunks and remember to commit early and often. But don't commit just to have something in the repository before leaving the office at the end of the day. If you're tempted to commit just because you need a clean working copy (to check out a branch, pull in changes, etc.) consider using Git's «Stash» feature instead.

WRITE GOOD COMMIT MESSAGES

Begin your message with a short summary of your changes (up to 50 characters as a guideline). Separate it from the following body by including a blank line. The body of your message should provide detailed answers to the following questions:

- › What was the motivation for the change?
- › How does it differ from the previous implementation?

Use the imperative, present tense («change», not «changed» or «changes») to be consistent with generated messages from commands like git merge.

VERSION CONTROL IS NOT A BACKUP SYSTEM

Having your files backed up on a remote server is a nice side effect of having a version control system. But you should not use your VCS like it was a backup system. When doing version control, you should pay attention to committing semantically (see «related changes») - you shouldn't just cram in files.

AGRE

Git lets flows: |
ches, n
you ch
your pr
deploy
import
person
work, j
workflo

HELP

Get hel

\$ ait

http:

http:

30-day free trial available at
www.git-tower.com

