Multiple Versions of the Present

Revision control tools such as Git not only let you return to past versions, but also let you keep multiple different "present" versions, via a feature called *branches*. To see why you might want multiple "present" versions, imagine that you are developing a large software project and have decided to begin adding a new, complex feature. Adding this feature will take you and your team of five developers four weeks to complete. One week into the effort, a critical problem is found in the currently released version of your software, which must be fixed as soon as possible.

Such a situation is a great use of branches. The development team might maintain one branch, **production**, which contains code that is ready for release. The only changes that may be made to the production branch are those that are ready for deployment. Another branch, **development**, can be used to work on active development—adding new features, testing them, etc. In the situation described above, the development branch is where the team would be adding the new complex feature.

How do these branches help? The developer assigned to fix the bug in the production code can *checkout* that branch (working on that version of the "present"), while the other developers can continue working on the development branch. In fact, this developer could (and should) make a new branch (let us call it **bugfix**) to work on the bug fix. The developer can then commit changes to bugfix, and other developers can switch to this branch if they need to work on that fix as well. When the bug fix is complete, it can be merged back into the production branch, incorporating the changes into the production software.

At the same time, development of new features continues on the development branch. When the bug fix is completed, it can also be merged into the development branch. Likewise, when the development of new features are completed and ready for release, they can be merged back into the production branch. Whenever these merges happen, Git will perform similar actions as in the case of merging in changes pulled from another repository—either handling it automatically if it can, or requiring the developer to figure out how to resolve conflicts.

Note that this is a bit of a small-scale motivational example. If we were really developing a large piece of software, we would almost certainly want more branches than those described above. We may have different developers working on different features in parallel, and want to manage combination of those changes with branches. We might also want branches for different stages of testing, or other purposes.

Completed