Advantages of DVCS

* huge in open-source
* in dvcs each system is a version control system unlike the cvcs where only the centralized server is one.
* Branching architecture is different in both,working directory changes with change in the branch checked out in case of dvcs whereas a folder exists for each new branch in case of cvcs.
* In dvcs changes can be made and committed in the clones and only pushed(or merged after review) to the remote repository.
* Context switching using branching is simple and quick.  This plays well in the case of working on bug fixes and developing new features in a software development context.  Being distributed, branching allows for individuals to create their own personal source branches which are not readily accessible by others.

Advantages of DVCS over CVCS

(Atlassian-blog)

* Everything except pushing and pulling can be done without an Internet connection.. So you can work on a plane, and you won’t be forced to commit several bugfixes as one big changeset
* Performing actions other than pushing and pulling change-sets is extremely fast because DVCS only needs to access the hard drive, not a remote server.
* Committing new change-sets can be done locally without anyone else seeing them. Once you have a group of change-sets ready, they can all be pushed at once.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

Disadvantages of DVCS over CVCS

(Atlassian-blog)

* If a project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
* If a project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

Problems solved by DVCS

|  |  |
| --- | --- |
| Problem in cvcs | Solved by dvcs |
| Single point of failure. | Each client connected to remote repository is a version control system and can perform very well as a server in case of failure of the remote server. |
| Remote commits are slow. | Introduced concept of local commits: no network, wicked fast. |
| Unsolicited changes that may break your build. | Developer pushes their changes continuously.  Integrators review the code changes and can bring about changes to the reference code asynchronously. |
| Merging is painful. | Merging is breeze. |

Comparative study

In due course of our evaluation, we found that subversion is versatile and a feature rich tool which user friendly tools and interfaces. This would be an automatic choice for project development teams which work in a closely knit work environment. It has a small learning curve and a new user can be brought up to speed very quickly. But on the down side, if the project involves a lot Merging and Conflict in future, it might be difficult to manage them when volume increases.

In this case, git has an upper hand. Git has a lot of handy features, very effective in exposing merge conflicts and managing multiple parallel versions. Although it has a steep learning curve, it is very versatile and works well in a dynamic project environment. Open Source projects often look towards Git as an effective VCS due to its versatility. Most projects these days are moving away from the Centralized approach to the Distributed approach for the same reasons. Based on our experience and findings so far, it is our opinion that Distributed approach outweighs the pros and outdoes the cons of Centralized approach. Hence, we feel that Git is the way to go moving forward.

Key advantage of DVCS

With distributed, the authoritative or central source is the source you want it to be, rather than being constrained by the system into having to have your source in one place.  
There have been occasions where we have had to use one of the developer’s local repositories when the central server has been down.  
This is simply making a decision that the entire team is aware of, that you are going to push / pull to / from an alternative repository.  
Hg has it’s own inbuilt web server, so this is very easy to do.

Scenario of better cure from data loss

While there is a slight danger a users hard drive could crash before he pushes his changes, that danger exists with a centralized system too.

However, the centralized system has a far worse disaster scenario. What if that “IT-guaranteed backup” of your subversion server you’ve been so confident in turns out to be corrupted? What if the backup job was modified by some guy who was late for his daughter’s recital and didn’t quite get it done right?

With a DVCS, you have no loss of data whatsoever. Just figure out who has the most recent commit – they have a COMPLETE copy of the repository. Within minutes you can have another centralized repository setup with absolutely no loss of data.

But you say – the same thing is true of subversion! We all have a copy of the project on our hard drive.

No, you don’t. You all have a copy of one VERSION of your application. It may not even be working or complete. And all those previous versions? Log entries? Metadata? Gone. Unless your backup is good, you’re hosed.

If fault-tolerance is crucial to your organization, your safest bet is DVCS with a backed-up shared repository. That way, you’re completely covered even if a backup should fail.

Pushing changes more often or not???

if you don’t push/pull changes to at least one other machine, local HDD failure results in loss of work – so you either need to push every time – and lose the benefit of not having to be online – or you have to accept that it’s more likely you’ll lose work.

Scenario

The workflow that you are describing above should not be a problem using DVCS (nor should it be the case with any type of VCS). Version Control Systems are design to help you and your team to collaborate actively on the same file. Because, as you grow, the chances of seeing different people hitting the same files get higher and higher  
Unwanted modifications can be avoided with a simple review process. If you make sure that someone looks at the new changesets you are probably going to catch most of the mistakes early in your development workflow.  
Regarding the local copy of the repository that exists in DVCS you just have to synch it regularly with your main repository to stay up-to-date. It’s actually as easy as pulling the latest version of the code.  
Finally, on the internet connection part we all need one when it comes to working together as a team 

Left to explore

<https://www.youtube.com/watch?v=4fsSyLkBdB4#action=share>

<http://programmers.stackexchange.com/questions/134514/moving-from-a-cvcs-to-a-dvcs-in-a-big-company-what-is-the-right-way-to-do-it>

<http://stackoverflow.com/questions/111031/comparison-between-centralized-and-distributed-version-control-systems>

<http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/>

<https://www.appfusions.com/display/StashSCMImporter/CVCS+vs.+DVCS+In+a+Nutshell>

<http://chriswongdevblog.blogspot.in/2010/11/dvcs-vs-subversion-smackdown-round-3.html>