

Goals:

What it means to replace CVS

What it means to roll our own Git-powered project hosting

What if we just used GitHub, or some other third-party project host?

PM Direction

Community Involvement

AMI

Documentation

Git Deploy

Code Additions, Changes, and Refactoring

VCAPAPI (& Git backend)

Refactor VC Project

Git - Specific Tasks and Infrastructure

Migration Process - three phases

git.drupal.org - Creation and Update

Job Queueing System

The Gist

SAM. PLEASE FILL ME OUT. :)

Background

What it means to migrate Drupal.org from CVS to Git

Source code migration

Project Hosting

Access Control

Facilitating Community Involvement

Alternatives to "Rolling Our Own"

dfkjsdhfjsdhfkjshdkafOpen Issues - "Things we aren't sure on"

Multiple Email Management / Commit Statistics

Project ACLs

User ACLs / Branch ACLs

Drush Scripts

Examples of Crazyiness:

PM Direction

Community Involvement

AMI

Documentation

Git Deploy

Code Additions, Changes, and Refactoring

VCAPAPI (& Git backend)

Refactor VC Project

Git - Specific Tasks and Infrastructure

Migration Process - three phases

git.drupal.org - Creation and Update

Job Queueing System

[Open Issues - "Things we aren't sure on"](#)
[Multiple Email Management / Commit Statistics](#)
[Project ACLs](#)
[User ACLs / Branch ACLs](#)
[Drush Scripts](#)

Goals:

- Provide Chris the information necessary that he can do a high-level overview which can justify a 1k hour project.
- Come up with a PM system that meshes with the DA's needs and Sam's sanity (need Kieran's help for this, and will be decided on by the end of the week)
- Three paragraphs:
 - What it means to replace CVS
 - What it means to roll our own github-like web UI (e.g., a gitosis-like clone)
 - What if we just USED github, or some other third-party project host?

Status update - what's been done so far

What makes this project so complex

What it means to replace CVS

Migrating our source code basically means making everything at <http://drupalcode.org/viewvc/drupal/> show up at <http://git.drupal.org/>, exactly as it originally existed. Sounds pretty simple, on the surface. Here's why it isn't:

- Because of the variety of experience levels of our contributors, there are some "interesting" things in our CVS repository that represent migration challenges when using off-the-shelf scripts.
- Code changes are required to Update Status and PIFR (test bot), which needs to be able to read new tag/branch information.
- Managing all these git projects (vs. the single CVS repository) necessitates

- infrastructural changes and additions, such as the introduction of a job queuing system.
- Access control is needed around who can commit to what projects.

Access control is challenging, because of two factors:

1. Git has an identification layer built into the commits themselves (every commit has a full name + e-mail address that may or may not match a user's Drupal.org e-mail address) that indicates authorship. This authorship layer is entirely separate from the authentication layer, whereas in CVS authentication and authorship are one and the same. Consequently we must design new systems that map drupal.org usernames/accounts with authentication data.
2. The most widely-supported method of authenticating to Git is via SSH public keys, which presents a tremendous learning curve to our contributors, particularly if they're on Windows. Doing username/password authentication over HTTPS instead is possible, but that means writing our own infrastructure tools to deal with it, since there is a lack of mature options out there.

What it means to roll our own Git-powered project hosting

Our project hosting infrastructure entails the Project suite of modules: Project (for the project pages themselves, such as <http://drupal.org/project/drupal>, search integration, maintainer/co-maintainer management), Project Release module (which allows maintainers to create release nodes that auto-generate tarballs for the actual download links), and Project Issue Tracking module (the issue tracker). Several aspects of this infrastructure tie directly to CVS via the CVSlog module, which also provides things like committer statistics and views of commit logs (<http://drupal.org/cvs>), as well as code tracking integration on Drupal.org user profiles.

As the name might imply, CVSlog module makes the assumption that the underlying version control system for the project hosting infrastructure is CVS. This fact has severely hampered the general-use of these tools outside of Drupal.org, which in turn influences the lack of community-driven work in this area. Part of the migration involves swapping out this CVS-specific back-end with the Version Control API module, which can be used on top of (potentially) any version control system, vastly increasing our chances of adoption of this suite of tools outside of Drupal.org.

Completing this work entails the following:

- Complete the 6.x-2.x branch of Versioncontrol API module (<http://drupal.org/project/versioncontrol>) and Versioncontrol Git module (http://drupal.org/project/versioncontrol_git).
- Complete the 6.x-2.x branch of Versioncontrol / Project Module integration module (http://drupal.org/project/versioncontrol_project). A considerable portion of this work is currently the responsibility of 3281d.
- Write Git hooks to update the log viewer and committer statistics views.

The commit statistics issue is particularly thorny, owing primarily to the split between commit authorship and network authentication discussed above. That authorship and authentication went hand-in-hand in CVS made statistics fairly straightforward: for every file you send in, you

get a commit. In git, however, authentication is irrelevant to authorship, so every time someone sends in new commits (a 'push'), we need to parse each commit for authorship and, if possible, assign credit.

These pairings between commits and d.o user accounts will also need to be periodically refreshed, as it is quite possible for a commit by an unknown author (i.e., unrecognized email address) could be sent in, then that email address is later added to d.o, and credit should be retroactively assigned.

There are further challenges in this area, as well, but these are the most immediately pressing ones. Given that commit activity is one of the few aggregate statistical measures we provide about d.o contributors, and that employers already use it for hiring decisions, we think it very important we really get these *right*. Moreover, proper credit for contributions was one of the major benefits originally cited in the discussion to move to git.

What if we just used GitHub, or some other third-party project host?

An understandable question that may be raised when confronted with the seemingly daunting (not to mention expensive) task of re-tooling all of our existing hosting infrastructure to use Git might be "Well, why not just have Drupal use GitHub?", or a self-hosted option such as Gitorious.

There are certainly pros to this approach. Particularly if we completely "out-sourced" to GitHub:

- Off-set all of our infrastructure requirements for code hosting to someone who specializes in how to do huge hosted code repositories. This would ultimately reduce costs, in both money and personnel, associated with our hosting infrastructure.
- We'd be able to perform faster upgrades of Drupal.org to major core versions, because we would not have to upgrade all of the various project hosting baggage each time.
- Large gain in features: nicer source code browsing, the ability to "follow" people, UI-based 'patch' merging, etc.
- Many contributors have already moved to GitHub, and we ultimately can't compete with their offerings, feature-per-feature (nor should we try; we have a CMS/CMF to build :)).

However, there are some fairly large cons as well:

- Rather than just having to re-train our code contributors on how to manage releases and patches, we now have to retrain **absolutely everyone**, including the folks who want to download a module or theme, or people who want to file a bug or support request.
- We also lose some features: issue tagging, the ability to move an issue from one project to the other if it's filed in the wrong place, integration into drupal.org user profiles, etc.
- Instead of a heavy new development burden, we would instead require a heavy migration/integration code burden: all existing issues, commits, etc. would need to be migrated, search results integrated, user accounts integrated, integration of Update Status and the testing bot, and so on. *Out-sourcing doesn't come for free.*
- The "not invented here" syndrome that causes us to write our own project/code management tools actually has many benefits for the Drupal project: seamless integration with our website (after all the work on Drupal.org redesign, it would be

a shame to lose this), numerous “upstream” patches to Views, Drupal core, etc. to facilitate advanced use cases, the ability to extend functionality however we see fit, since it’s using our framework.

- There was a lack of community support for this option at the time the CVS -> ? discussion was held back in February: <http://groups.drupal.org/node/48818> Particularly for closed-source options such as GitHub.
- Abandoning ship at this point means starting the very substantial discovery process from scratch all over again.
- No way to enforce cultural/legal requirements, e.g. all code must be GPLv2+. Becomes “wild west.”
- Most importantly, we lose Drupal.org as our central development hub. Drupal.org instead becomes a support forum and marketing platform for the Drupal project. This has pretty severe community cultural implications: Developers are off on GitHub, end users are on Drupal.org. Much fewer smart people answering end users questions, and little to no user-to-contributor cross-over. Very little developer cross-pollination as well; everyone works in their own sandbox.

Basically, this is definitely an option we should talk about, but definitely not a decision to make lightly.

Detailed Implementation Plan & Estimates

PM Direction

It's either Chris, or some other PM gets hired, or we find a volunteer from the community. Sam's #1 preference would be that Chris gets 20hrs/wk or so and we can do this, because this relationship is clearly working.

Community Involvement

AMI

Tasks:

- Set up migration process hudson job
- Tweak existing git-dev build job to actually create an AMI (how to do this, i dunno)
- Secure some dolla dolla bills for AMI resources

- Getting vcapi stable (e.g., finishing the major refactor & at least log parsing)
- Update vc_project to new vcapi
- Some amount of git-server-specific configuration (setup of HTTP/SSH? determine fs arrangement/perms?)
- Plus, this is just plain settin up a server!

Documentation

- Front page d.o post RE: Git and ways to get involved
- Updates to the community initiatives page
 - Includes general updates to the "what Git is" doc
 - Talk specifically about involvement
 - Consolidate a list of issues that the community can get involved in.
- Logging of new "shovel ready" issues
- Come up with a tagging system for issues
- Periodic status updates to inform the community of progress

Git Deploy

Equivalent of CVS Deploy, for sites that are deploying their modules/themes from Drupal.org's Git Repository.

Code Additions, Changes, and Refactoring

VCAPI (& Git backend)

- Finish 2.x branch, including addition of dbtng and entities (more maintainable code, easily ported to D7)
- Make admin UI views-based
- Update commitlog & its blocks to views
- Update log parser, get it working flawlessly
- Git hooks for populating the db on push
- Whole new approach to commit statistics

Refactor VC_Project

- UI-based restrictions on branches/tags
- Catch back up to vcapi 2.x branch
- Shift branch/tag restrictions into UI filter
- Release node integration (3281d)
- Co-maintainer designation (3281d)
- Need to investigate more to determine additional TODOs

Git - Specific Tasks and Infrastructure

Migration Process - three phases

- 1:1 data parity - all your branches/tags (and ONLY those branches/tags) are there, with all the right data, exactly as it was in CVS
- Handling non-optional data transformations: mapping of cvs usernames to new data (which has to be collected); transforming branch/tag names.
- Post-process transformations, such as recursive stripping out of \$Id\$ tags.

git.drupal.org - Creation and Update

- Set the migration script (<http://github.com/sdboyer/drupalorg-git>) up as a hudson job to regularly rebuild the git.drupal.org data

Job Queueing System

- We need a jobqueue system that will initially be used to create repositories on-disk (so that the webserver need not have direct write access to repos), but would eventually be used for many more operations. The infrastructure team has been consulted on this, and agrees that such a system is necessary, and that it would also be quite beneficial for some other systems (packaging, for example, could benefit). The current plan is to use beanstalk.

Asking for cash, justifying what additional cash would help with.

//// What follows is webchick's "The Drupal.org Git Migration Effort for the Non-Initiated." I'm not sure if there's value in this or not; it might be a blog post on a.d.o. But I feel like both the community and the DA could use some real context in order to swallow the sudden huge price increase of this initiative.

The Gist

We're now about 8 weeks into this initiative. This time has focused on the following items:

- Discovery & Planning
- Communication & Volunteer coordination
- Versioncontrol API Refactoring

The requirements for this project were only partially understood during the original brainstorming session that took place at Drupalcon San Francisco. If you look at the [original community initiative page](#), it is heavily weighted towards top-level tasks that were obvious from the 'outside.' We did a good job identifying most of the major areas of work - really, versioncontrol* is the only area that emerged as requiring significant time investment after the contract started - though at the beginning, no one had a firm grasp of how much work each of the areas would entail, and especially the work required to combine everything together. Consequently, discovery was a major part of the initial work - each of the major areas needed to be stepped through to figure out what they *really* entailed, and where the dependencies were on the other areas, and on the other DA contractors. In June, at the very beginning, the primary focus was just discovery. By early July, we had moved on to proper project management and planning, now largely complete; much of what is expressed in this document reflects that planning process.

It's always been our goal to involve the community as much as reasonably possible; from the outset, we recognized the migration as both being impossible to complete without volunteer help, but also as an opportunity to grow the circle of people capable of contributing to d.o's project architecture. There's been a fair degree of success with this so far - one volunteer is primarily responsible for working on the migration scripts, another is taking the lead on versioncontrol's Views integration, and there is an assorted half-dozen who've contributed reviews and some patches, primarily to versioncontrol. However, it's also become clear that the time required to bring coding volunteers up to speed is prohibitive for a paid contractor - much of the reason why we are emphasizing creating an AMI, which will lower barriers to volunteer entry.

The bulk of actual development work has been on Versioncontrol API. At the beginning of the contract, versioncontrol's 2.x branch (a major OO refactor undertaken by marvil07 for his 2009 Google SoC project) was approaching stability, but was incomplete, and had not addressed some of the flaws in the original version. After a thorough evaluation, it was determined that rather than simply move towards stability, we should take the extra time to remove the problematic architecture and replace it with a forward-looking, easier-to-maintain code (summary: move from hacky custom loaders and procedural code copy/pasted into OO over to using DBTNG, a partial backport of D7 entities, and a saner class hierarchy). The ultimate goal here is to use some of the paid work time to avoid perpetuating the project* problem: it might be stable, but it's unwieldy, crufty, and virtually impenetrable by anyone who doesn't already know it. While this has often taken a backseat to the other considerations described above, it's been the primary work area whenever time has allowed.

The good news is we have made tremendous progress, both in necessary development work required to perform the migration, and moreover in ferreting out what we believe are all of the sub-initiatives necessary to complete in order to achieve a successful migration. This was no easy task, as the depth and breadth of the this migration project is incredibly vast, the extent

of which couldn't possibly be known until much time was spent determining what all the pieces were.

The bad news is, now that these pieces are clear, this is a *much*, much more massive project than the Drupal Association budget had originally been scoped for to put them all together. And while we originally had high hopes about "crowd sourcing" much of the development tasks, and Sam is still actively seeking out those opportunities and matching them with volunteers, it's clear now that much of this work can only be done by a handful of experts in the community.

// Something something.

Background

For many years now, Drupal.org's use of CVS as a version control system has been severely debilitating to its contributor community. CVS is an archaic technology long surpassed by superior systems, and is today only used in legacy projects like 10-year old CMS projects. ;) This means that nearly all contributors coming to Drupal have an extremely steep learning curve to surmount in order to contribute new code to the project, forced to learn arcane skills that only apply when doing Drupal-related work. This long-time annoyance is gradually heading to a crisis point, as with the advent of more sophisticated code hosting platforms on modern version control systems that developers already use in their day-to-day lives (e.g. Launchpad and GitHub), many contributors (and many of these *key* contributors), frustrated with Drupal.org's use of CVS, are moving their code off of Drupal.org and onto these sorts of alternative repositories, or simply contributing back nothing at all because the learning curve is too steep.

The by-product of this is that the Drupal community is in genuine peril of losing its centralized code repository, and thus its centralized development hub, from which the project gains many important benefits: a one-stop shop for end-users to find all Drupal-related code, legal clarity on all code from d.o being GPLv2+, a single set of tools (suboptimal as they are) for contributors to use that work exactly the same across *all* projects, peer code reviews from others involved in the Drupal project, and a closely-knit developer culture that encourages mentorship and collaboration.

The debate reached a fever pitch in February 2010, when the Drupal community decided to once and for all figure out what to do about the CVS problem at <http://groups.drupal.org/node/48818>. The result of the unprecedented 400+ reply discussion was to move to Git. The overall reasoning was that Git has the largest support in the Drupal community (which means lots of people for Git newbies struggling to figure it out to ask for help), seemingly the largest support *outside* the Drupal community as well, the most pre-written code of any other version control in terms of integration with our infrastructure, an eager team of volunteers willing to help with the migration, and, as a distributed version control system, an architecture that supports the way our community works.

At Dries's keynote in Drupalcon SF in March 2010, he announced that Drupal.org was moving to Git, to wild applause (and also expectation of action/execution). At the Drupal Association retreat following Drupalcon SF, the Drupal Association collectively agreed to the CVS to Git

migration being one of our top priorities for 2010. Budget was allocated for a Git Migration Lead contract, interviews were held with several candidates, and Sam Boyer (who was key in developing the implementation plan outlined at <http://drupal.org/community-initiatives/git>, and has a solid technical grasp in the various areas of the migration) was tapped for the position in June 2010.

What it means to migrate Drupal.org from CVS to Git

To the uninitiated, the task of migrating from CVS to Git probably sounds fairly straight-forward. After all, a quick web search can reveal several pre-existing scripts that claim to do the task, and there are also several other open source projects that have done the migration already who we can ask for help. So what's the big deal?

The following outlines the general areas of the migration, what has been accomplished so far, and what is left to do.

Source code migration

The most obvious task that needs to be performed in a CVS to Git migration is actual migration of the source code from one system to the other. While pre-existing scripts for doing this migration do exist, these scripts result in weird inconsistencies when run on Drupal.org's data. Presumably because, like most import scripts, once they run well enough for someone to do what they need to do, they have no reason to back and further tweak them. :)

Most of these conversion issues have now been worked out, and there is a set of scripts at <http://github.com/sdboyer/drupalorg-git> that do a near-complete translation on our data. You can see the results at <http://git.drupal.org/>, a read-only mirror of the CVS repository. This is an important milestone, because documentation, project integration, and other dependent items require this Git clone in order to proceed.

Remaining items:

- Verifying 1:1 data parity between the CVS and Git mirror; ensure all branches/tags (and ONLY those branches/tags) are there, with all the right data, exactly as it was in CVS.
- Transforming branches/tags to new "2.x-1.0" / "2.x-1.x" format.
- Map CVS usernames to Git equivalent. This is new data, which has to be collected.
- Optional transformations, such as recursive stripping out of \$Id\$ tags.
- Set the migration script up as an automated Hudson job to regularly rebuild the git.drupal.org data so we can continually test the migration until we pull the final trigger.

Project Hosting

Where the Drupal project differs from a number of other open source projects that have made this switch is we effectively run our own community-specific SourceForge.net: over 6,000 projects by over 3,000 individuals are hosted on Drupal.org. This centralization in turn provides our project with a vibrant, collaborative development community and a "one-stop shop" for our end users to find any code they need to extend their Drupal site. A huge benefit is that once a contributor learns the toolset for contributing to some aspect of Drupal (a theme, a module),

they can re-use that same set of skills for *any* code in the project. This is a unique facet of our community, and one of our key strengths.

Something else that's unique about our project hosting is the breadth of experience levels among its contributors. Everyone from hard-core Drupal core developers who can code PHP in their sleep, to hobbyists who are building Drupal sites for their church, are using drupal.org to commit and download code, exchange and test patches, and collaborate on solutions to problems. As a result, the learning curve associated with this migration, particularly for non-developers, is of tantamount importance so we do not cull our contributor base in the process. This makes the migration of the project hosting part of Drupal.org particularly challenging.

The Project hosting components of Drupal.org effectively boil down to the following aspects:

1. The project release system, which generates the module/theme download tarballs based on tags/branches in the version control system. Currently provided by Project Release module and CVS log module, plus some server-side scripts.
2. Contributor tracking and statistics: things like the list of a project's maintainers, lists of commit messages, etc. Currently provided by CVS log module.
3. Some miscellaneous add-on functionality, such as the testing bot "CVS instructions" tab on project pages, which offers copy/paste commands to help ease the learning curve for new users.
4. Commit restrictions, branch/tag restrictions, and assorted other server-side scripts that somewhat differentiate our CVS server from a vanilla CVS server.

There is also a variety of dependent infrastructure including the QA test bot and the CVS Deploy module.

As the name "CVS log" might imply, our current project hosting infrastructure is innately tied to CVS. All of this code needs to be migrated. This CVS lock-in is also a major reason for the lack of up-take in Project module in sites outside of Drupal.org: as mentioned above, CVS is an older system only used for legacy support.

The result of a Google Summer of Code initiative, the Version Control API module, and its sister projects Version Control Git and Version Control Project, abstract out the functionality provided by CVS log module so that it can be used across any version control system. One of the first development initiatives undertaken by Sam Boyer was some necessary refactoring

Going forward, our plan for the initial migration is to finish refactoring

- 5.
6. release system, which b
7. Tracking commit information, statistics, ,
8. Project Release module, part of the Project module, which handles
9. Project / Version Control integration: Currently this part of the system is tied to CVS
- 10.

Access Control

Drupal.org offers a sophisticated level of access control around its projects. Access to commit to the overall repository is limited to only users who have passed through an application process. Each project has an access control list (ACL) which restricts commit access to only a small subset of people: a “maintainer” and potentially one or more “co-maintainers”. This ensures that webchick can’t accidentally commit stupid code to Views module, and that each project “lead” has the ability to maintain the keys to who can make changes to their module or theme. In order to commit, you must authenticate yourself to the system, and an ACL is checked to ensure you’re allowed to do what you’re asking.

A key component of access control is also the concept of identity: who owns the project, whose patch was committed, who made the commit. This helps build each contributor’s reputation, and in turn helps build a level of familiarity among other developers. When the nickname “merlinofchaos” is seen next to a project or a commit, it carries with it an understanding of the level of quality therein.

Translating these concepts from CVS to Git is proving very challenging. Two key fundamentals are:

- **Authentication:** In CVS, one authenticates against the server with a username and password (passed over the wire in plaintext). In Git, this is typically done with public SSH key authentication, for which access control scripts already exist that we could use, is far superior from a security point of view. However, this presents an *enormous* learning cliff for a large portion of our contributor community, especially for Windows users. Another option is HTTPS, which would use usernames and passwords as contributors are used to, but is a relatively new option in the Git world and so would require us to write our own ACL scripts.
- **Identity:** In CVS, all commits, commit messages, and so on refer back to a single, canonical drupal.org username. In Git, identity works differently: a global config file maps your identity credentials (usually full name and e-mail) to a commit. This e-mail may or may not map to the e-mail we have on file for the user in the users table. We need to build a method of associating multiple e-mail addresses with a single account, so that regardless of which computer Earl Miles commits from, he is properly credited by the system.

This is still an area where there are many unknowns.

Facilitating Community Involvement

A major move like this only ends successfully when we have large buy-in from our contributor base, an enormous support network of resources (both written/visual and human) for those struggling to make the transition, and easy jump-off points for eager volunteers to help aid in the implementation efforts.

Part of this comes down to a communication strategy, which is where the Drupal.org Git

Migration Team working group (<http://groups.drupal.org/drupal-org-git-migration-team>) and Twitter announcements via <http://twitter.com/DrupalGitGremlin> come in: these have been set up in order to communicate major milestones, and calls for action in areas of the project that need assistance. Another huge piece is an open and transparent method of working that encourages community participation, which is why all parts of the Git migration initiative are tracked in the public Drupal.org issue queue under the “git phase 2” tag: http://drupal.org/project/issues/search?issue_tags=git+phase+2

Sam also kicked off an Open Source Training Curriculum initiative (<http://groups.drupal.org/node/80539>), which aims to crowd-source the considerable documentation/training efforts around the migration into a re-usable curriculum package that can be taught at Drupalcon, DrupalCamps, and local user group meetups.

However, while training, documentation, and to some extent decision-making tasks are relatively easy to crowd-source, much of the heavy lifting on this project relates to highly specialized development tasks. It’s difficult to find volunteers capable of contributing skills-wise, and then given the added complexity of the Drupal.org infrastructure, it becomes almost impossible to set them up for success. A contributor new to Drupal.org’s infrastructure invariably spends hours of both their own and an infrastructure team representative’s time getting all of the prerequisites, coming up to speed on the myriad of technologies used on d.o, and navigating all of the other various hurdles. The Drupal.org Redesign project faces similar challenges, although the source code migration aspect with Git makes this particularly challenging for our team.

Going forward, we feel that there is significant benefit to investing time and infrastructure into setting up automated sandbox environments based on Amazon Machine Interface (AMI), where the workflow for new contributors getting involved would change from “kiss off a weekend or two” to “click a button to spin up your own copy of git-dev.drupal.org” which would have on it an already set-up copy of the Git mirror, a full copy of Drupal.org with sanitized database in place, and all development code from other team members. All volunteers would need to do is log in and get started. This would allow us to not only dramatically increase our ability to crowd source aspects of the project beyond docs and decisions, but also re-use this same technique to enable volunteers to help on other d.o properties (qa.drupal.org, Drupal.org redesign, etc.).

Remaining items:

AMI

- Secure funding for AMI resources
- Tweak existing git-dev build job to actually create an AMI (requires research)
- Some amount of Git-server-specific configuration: set-up of HTTPS/SSH, determine fs
- Additional server set-up
- Testing and QA

Documentation

-
- - Getting vcapi stable (e.g., finishing the major refactor & at least log parsing)
- - Update vc_project to new vcapi
- - Some amount of git-server-specific configuration (setup of HTTP/SSH? determine fs

• arrangement/perms?)

Alternatives to “Rolling Our Own”

An understandable question that may be raised when confronted with the seemingly daunting (not to mention expensive) task of re-tooling all of our existing hosting infrastructure to use Git might be “Well, why not just have Drupal use GitHub?”

dfkjsdhfjsdhfkjshdkaf

Open Issues - "Things we aren't sure on"

Multiple Email Management / Commit Statistics

- Properly implementing good, reliable commit statistics will eventually necessitate allowing people to register multiple email addresses on their d.o accounts. We don't need it for launch necessarily, but will need it eventually.

Project ACLs

- Infrastructure
- Choosing which method we go with: SSH or HTTPS

User ACLs / Branch ACLs

- Do we need this?

Drush Scripts

- Not necessary for migration, but make learning curve much easierlack of community buy-in for this option, or at least was at the time the CVS => ? migration was initially discussed:

Examples of Crazyiness:

- Need a whole system for managing writes on the server - previously, contributors created new directories in CVS directly via CVS commit commands. No longer possible, as there are separate repos for each project - so we need a system that can create new, empty repositories for projects that people can push into.

PM Direction

It's either Chris, or some other PM gets hired, or we find a volunteer from the community. Sam's #1 preference would be that Chris gets 20hrs/wk or so and we can do this, because this relationship is clearly working.

Community Involvement

AMI

Tasks:

- Set up migration process hudson job
- Tweak existing git-dev build job to actually create an AMI (how to do this, i dunno)
- Secure some dolla dolla bills for AMI resources
- Getting vcapi stable (e.g., finishing the major refactor & at least log parsing)
- Update vc_project to new vcapi
- Some amount of git-server-specific configuration (setup of HTTP/SSH? determine fs arrangement/perms?)
- Plus, this is just plain settin up a server!

Documentation

- Front page d.o post RE: Git and ways to get involved
- Updates to the community initiatives page
 - Includes general updates to the "what Git is" doc
 - Talk specifically about involvement
 - Consolidate a list of issues that the community can get involved in.
- Logging of new "shovel ready" issues
- Come up with a tagging system for issues
- Periodic status updates to inform the community of progress

Git Deploy

Equivalent of CVS Deploy, for sites that are deploying their modules/themes from Drupal.org's Git Repository.

Code Additions, Changes, and Refactoring

VCAPAPI (& Git backend)

- Finish 2.x branch, including addition of dbtng and entities (more maintainable code, easily ported to D7)
- Make admin UI views-based
- Update commitlog & its blocks to views

- Update log parser, get it working *_flawlessly_*
- Git hooks for populating the db on push
- Whole new approach to commit statistics

Refactor VC_Project

Git - Specific Tasks and Infrastructure

Migration Process - three phases

- 1:1 data parity - all your branches/tags (and ONLY those branches/tags) are there, with all the right data, exactly as it was in CVS
- Handling during-the-migration data transformations: mapping of cvs usernames to new data (which has to be collected); transforming branch/tag names.
- Post-migration transformations, such as recursive stripping out of \$Id\$ tags.

git.drupal.org - Creation and Update

- Set the migration script (<http://github.com/sdboyer/drupalorg-git>) up as a hudson job to regularly rebuild the git.drupal.org data

Job Queuing System

- We need a jobqueue system that will initially be used to create repositories on-disk (so that the webserver need not have direct write access to repos), but could eventually be used for many more operations (spawning per-issue repos, generating tarballs on-request), and could also be used by existing systems (packaging script). Hudson is probably not reusable for this; we're looking for a tool more in the class of what's discussed here: <http://github.com/blog/542-introducing-resque> . An alternative backend to DrupalQueue has been suggested.

Open Issues - "Things we aren't sure on"

Multiple Email Management / Commit Statistics

- Properly implementing good, reliable commit statistics will eventually necessitate allowing people to register multiple email addresses on their d.o accounts. We don't need it for launch necessarily, but will need it eventually.

Project ACLs

- Infrastructure

- Choosing which method we go with: SSH or HTTPS

User ACLs / Branch ACLs

- Do we need this?

Drush Scripts

- Not necessary for migration, but make learning curve much easier