Gitサーバー比較

【独断による結論として、最もお勧めな構成は、13.RhodeCode、次いで9.gitoliteもしくは11.GitLab
管理ツールを導入しない場合なら、3.SSH＋5.WebDAV(NTLM認証)の複合環境が扱い易い】

【前提】評価時のクライアント要件：	**OS：**	Windows 7/8
	使用ソフトウェア：	TortoiseGit1.8.3 + msysGit1.8.3
【前提】考慮すべき要件：		社内利用、閉鎖的な開発、ネット経由の安全な遠隔地開発、操作性、耐障害性（バックアップと迅速な復旧）、および、 ユーザーアクセス権限の管理（アクセスユーザーを限定する他、状況に応じて一時的に共有リポジトリへの書き込みを閉鎖、また、できれば過去バージョン用タグ・ブランチの更新を禁止にするなど）
【前提】注意点		各構成での注意点や一部気をつけるべき設定例などの記載はあるものの、完全な設定方法や手順は示さない為、基本的には書籍やネットの記事を参考に導入する必要あり。

【凡例】

良い評価は黄色に、	
悪い評価は赤色に、	
特に良くも悪くもなければ無色に、	
それが未評価項目の場合は灰色に、	…セルを着色

パターン No.	サーバー構成名	[1] プラットフォーム（OS）	[2] 評価に使用した OS・アプリケーションの バージョン	[3] リポジトリ アクセスの 通信方式	[4] ユーザー管理	[5] リポジトリ認証方式	[6] リポジトリアクセス時の パスワード／パスフレーズ指定方法	[7] 対応VCS	[8] セットアップ	[9] 社外からのアクセス	[10] 通信の安全性 ※機密性の保護
1	**Windowsファイル共有 (共有フォルダ)**	Windows ※Linuxも可（samba使用）	-	SMB/CIFS	Windowsユーザー／ ActiveDirectory	Windowsユーザー認証 （Active Directory）	自動。 ※Windowsのファイル共有依存	Git Subversion 他（未確認）	不要。	×	×
2	**Gitデーモン（サービス）** **※未評価**	Linux/Windows	-	Git専用プロトコル	なし	-	-	Git	簡単。 Git以外のインストール不要。 サーバー上で git daemon コマンドを実行するだけ。 別途ファイアウォールの調整も必要。	○	×
3	**SSH** **※Git使用時に最も標準とされる通信方式**	Linux ※Windowsは未評価(所用SSHサーバー他FreeSSHdなどを使用する方法が考えられる)	CentOS6.4 OpenSSH5.3p1-84 Git1.7.1-3	ssh	Linuxユーザー ※①クライアントのユーザー数分のLinuxユーザーを登録するか、②代表ユーザーを一つだけ用意する（例えばgitユーザー）。前者①ならLinuxのユーザーをWindows（ActiveDirectory）と統合した方が良いが、後者②の方が扱い易い。以降の説明は後者②を採択したものとして記述。	RSA（公開鍵認証）	サーバーアクセス時に毎回秘密鍵のパスフレーズをダイアログ入力 もしくは Pageant（TortoiseGit付属ツール）を使用して、パスフレーズを自動入力すると便利。 ※Pagenantでは、パスフレーズを入力した状態の秘密鍵が内部で管理され、サーバーの共通鍵にマッチする秘密鍵が設定済みなら、パスフレーズの入力を省略して秘密鍵が自動的に指定される。ただし、複数の共通鍵をサーバーに登録している場合、かつ、Pageantに複数の秘密鍵を登録した場合、自動的に鍵が選ばれてしまうので、同一クライアントからユーザーを切り替えて使うといった利用がしにくい（主にテスト時や管理者ユーザーと開発者ユーザーの二役をこなす場合など）。その場合、都度、Pageantから鍵を削除するか、Pageantを再起動して登録済みの鍵をクリアする。	Git	簡単。 ※sshとGitをインストールすればほぼ完了。 ※ただし、SSHのセキュリティを強固にする必要あり。（端末ソフトでログインして、ファイルを漁ったり、コマンド実行したりする事ができないようにする必要あり。専用シェルgit-shellを/etc/passwdのGit専用ユーザー(git)に対して指定する方法が一般的。） 　例：git:x:501:501:git controller:/var/lib/git:/bin/git-shell ※ユーザーのホームディレクトリには、sshの公開鍵を保管する為のファイルを用意する必要あり。~/.ssh/authorized_keys ファイル。パーミッションは、.sshディレクトリを 700 に、authorized_keys ファイルを 600 にしておかないと、正常に機能しない。（例：$ chmd 600 authorized_keys） ※authorize_keys ファイルの内容例：（１ユーザーに付き1行で公開鍵を記述する） ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA3zWAuDWLgCS3mDpJhRtfIUKcy8zp+Fx9hSWbXfemHpRfT/tAhV06eYwOplYNdyTXA/JNMq9AqMI8A2HossOeLM3W/3bNQeW9KhwRyR0sAFaVLKWAFY4oVWSamqeW92BV2hfnHA3jGT9DTFFpc9vBR7dfpsU2P8Hc6zSDvKNK6eF48VOrZKgmieJTPSnxH/Ui62daHSrHBG7XVuNBmM+/cs0O9t2Kbyuo5mydcz3uFBt9D7Q9xHXrNaa4/iBINFHjYQEm/xH4HgMfmZlay2WJc4DX1R4nP+egNFwJIjUcSOKDb9fACAbwKrVOiewpwyoXeBTao3V0rZSJDZAxLN2ZOw== user_a ssh-rsa	○	◎

パターンNo.	[1] サーバー構成名	[2] プラットフォーム（OS）	[3] 評価に使用したOS・アプリケーションのバージョン	[4] リポジトリアクセスの通信方式	[5] ユーザー管理	[6] リポジトリ認証方式	[7] リポジトリアクセス時のパスワード／パスフレーズ指定方法	[8] 対応VCS	[9] セットアップ	[10] 社外からのアクセス	[11] 通信の安全性※機密性の保護
4	**WebDAV（Basic／Digest認証）** **※http(s)通信で認証なしの場合も同様** **※Subversionでは最も一般的な構成（特にWindowsプラットフォーム）**	Linux ※Windowsは未評価	CentOS6.4 Apache2.2.15-28 mod_ssl2.2.15-28 samba3.6.9-151 Git1.7.1-3　※読み取り専用にするならgit不要 gitweb 1.7.1-3 ※必要に応じて（ブラウザ上からリポジトリを確認したい場合のみ）	http/https	専用のパスワードファイルに定義されたユーザー、もしくは、ページファイルに設定されたユーザー権限などを利用。	http(s)のBasic/Digest認証 もしくは、認証なし	【方法①】サーバーアクセス時毎回ダイアログ入力 【方法②】URLに直書き →（例）http://user:pass@server.domain/dir/ ※通常gitでは、Webブラウザからアクセスした時のように、もしくは、TortoiseSVNでアクセスした時のように、ユーザーとパスワードを自動記憶してはくれない。自動記憶させる方法もある⇒方法④。 ※プレーンテキストで書くので、gitの設定を見るとパスワード丸見え。 【方法③】ネット接続用設定ファイル C:¥Users¥(ユーザー名)¥_netrc に、対象マシン、ユーザー、パスワードwお記述 →（例） #machine server.domain #login user_a #password password-text ※プレーンテキストで書くので、ファイルを見るとパスワード丸見え。 【方法④】【推奨】（Ver.1.8.1から追加された仕様） Creditional HelperとしてWincred を使用し、TortoiseSVNのようにパスワードを記憶させる。 Creditinal Helperの設定： TortoiseGitの [設定]→[Git]-[Credential]画面で「資格情報ヘルパー」に「wincred - すべてのWindows利用者」を指定。 ※また、Tortoise Gitの [設定]→[Git]画面で[全ユーザ共通の設定を編集]を実行し、設定ファイルに下記のエントリーを追加してもOK。 [credential] helper = wincred ※【問題点】一度記憶されたパスワードをリセットする方法が不明！（TortoiseGit→[設定]→[保存されたデータ]→[認証データ]→[クリア]では消えない） →解決策：リセットするツールを自作。 （git-credential-wincred.c を元に作成） リポジトリ： https://github.com/gakimaru/public/tree/master/tools/git_erase_wincred_all git_erase_wincred_all.exe を実行すると、Git用に記録されているユーザーとパスワードのエントリーを一括削除する。確認なども出ないの	Git Subversion 他 ※Mercurialも可？ （未確認）	やや手間。 ※基本的には、Apacheをインストールし、WebDAVを有効にしてリポジトリのフォルダを公開すれば良い。 ※ユーザーが任意のリポジトリを作れるように、LocationMatchディレクティブを使用して設定すると良い。 ※リポジトリを読み取り専用にするなら、Gitのインストールは不要。 ※Push操作に対応するなら、Git をインストールし、git-http-backend スクリプトを有効にする。 （例） SetEnvIf Request_URI "/git/.*¥.git/.*" GIT_PROJECT_ROOT=/opt/git SetEnvIf Request_URI "/git/.*¥.git/.*" GIT_HTTP_EXPORT_ALL= ScriptAlias /git/ /usr/libexec/git-core/git-http-backend/ #Alias /git/ /opt/git/ <LocationMatch "/git/.*¥.git/.*"> … ※ApacheからWebサーバーへのアクセスで問題が生じないように、リポジトリファイルの所有者、アクセス権限を調整する必要あり。別の方法として、Apacheの設定を変更して、Apacheの実行ユーザーを Git 管理用のユーザーに変更するのも良い。 ※Subversionを使用する場合は、mod_dav_svn と mod_authz_svn が必要になるが、Gitにはそのような専用モジュールは不要。（Gitにはそもそもユーザー管理の仕組みがなどがその理由） なお、Subversionではこのモジュールさえあれば、Subverson自体のインストールは不要。 ※gitwebは別途yum install gitwebなどによりインストール可能な、Webサーバー向けのCGIスクリプト。 Apacheにも簡単に導入できる。 /var/www/gitweb.cgi 内の $projectroot = "…" の内容を書き換えてGit リポジトリのトップディレクトリを指定し、Apache上でgitweb.cgi を実行するように指定すれば、指定のディレクトリ以下のリポジトリの状態をWeb上で確認する事ができるようになる。 必須ではないが、あれば少しだけ便利。 やはりリポジトリ毎のユーザーアクセス権限の設定はできない。	○	○ ※https使用時（httpは危険）
5	**WebDAV(NTLM認証)**	Linux	CentOS6.4 Apache2.2.15-28 mod_ssl2.2.15-28 samba3.6.9-151 Git1.7.1-3　※読み取り専用にするならgit不要 mod_auth_ntlm_winbind　※2006年のソース？ gitweb 1.7.1-3 ※必要に応じて（ブラウザ上からリポジトリを確認したい場合のみ）	http/https	ActiveDirectory	ActiveDirectory統合認証 （NTLM/Kerberos）	（同上）	（同上）	やや手間。 ※WebDAVとGitの設定は上記Basic/Digestと同じ。 ※さほど難しくはないが、ApacheにNTLM認証やSSLの為のモジュールをインストールして、適切なApacheの設定を行う必要あり（モジュールの読み込みと認証方式の指定、認証サーバーの指定など）。 ※その前に、ActiveDirectory認証を使うために、Sambaをインストールし、ActiveDirectoryに参加（net ads jon）して、WinBindサービスを稼働させる必要あり。 ※その他、上記Basic/Digest認証と同じ。 ※Apacheの実行ユーザーが、Winbindのキャッシュファイルにアクセスできるようにする必要あり。ファイルの所有者を変更する方法が示される事が多いが、それだとApache以外の場面で問題を起こす事がある為、Winbindのグループに Apacheの実行ユーザーを追加する方法が無難。 （例）/etc/group wbpriv:x:88:apache,git	（同上）	（同上）

パターン No.	サーバー構成名	[1] プラットフォーム（OS）	[2] 評価に使用した OS・アプリケーションの バージョン	[3] リポジトリ アクセスの 通信方式	[4] ユーザー管理	[5] リポジトリ認証方式	[6] リポジトリアクセス時の パスワード／パスフレーズ指定方法	[7] 対応VCS	[8] セットアップ	[9] 社外からのアクセス	[10] 通信の安全性 ※機密性の保護
6	**WebDAV(SSPI認証)** **※検証不十分** **※Subversionでは一般的な構成の一つ**	Windows	Windows Server 2012 Apache2.2.24+SSL(32bit) mod_auth_sspi1.0.4-2.2.2(Apache2用) msysGit1.8.3 ※読み取り専用にするならgit不要 gitweb 1.7.1-3 ※必要に応じて（ブラウザ上からリポジトリを確認したい場合のみ）	http/https	ActiveDirectory	ActiveDirectory統合認証 （SPSPI/Kerberos）	（同上）	（同上）	やや手間。 ※WebDAVとGitの設定は上記Basic/Digestと同じ。 ※さほど難しくはないが、ApacheにSSPI認証やSSLの為のモジュールをインストールして、適切なApacheの設定を行う必要あり（モジュールの読み込みと認証方式の指定、認証サーバーの指定など）。 ※Linuxと異なり、ActiveDirectoryの参加やWinBindの稼働は不要。 ※その他、上記Basic/Digest認証と同じ。	（同上）	（同上）
以下、管理ツールを導入する構成											
7	**gitweb** **※リポジトリの状態確認用の補助ツール。WebDAV,SSH,gitosis,gitliteと組み合わせて使用する事が可能。プラットフォームやアプリケーション、セットアップ方法等の説明については上記WebDAVの構成を参**	-	-	-	-	-	-	-	-	-	-
8	**gitosis** **※未評価（後継ツールとも言うべきgitoliteのみ検証し、gitosisについては一切の調査**	-	-	-	-	-	-	-	-	-	-

9	**gitolite**	Linux
		※Windowsは未評価
			CentOS6.4
			OpenSSH5.3p1-84
			Git1.7.1-3
			perl5.10.1-131
			gitolite3.5 ※ソースからインストール
				ssh	gitolite動作・管理用のLinuxユーザー一つ（gitなど）
					＋
					gitolite独自ユーザー管理
						RSA（公開鍵認証）	上記SSHのケースと同じ。	Git	比較的簡単。
									
									Gitolite動作用の専用ユーザー（仮に git ユーザーなど）をLinux上に作成した上で、READMEに書かれた手順に従ってインストール。
									
									大まかな手順は下記の通り。
									①gitユーザー作成。（以下、gitユーザーでログインして操作）
									②~/.ssh/authorized_keys を用意（上記SSHのセットアップ参照）。$ touch ~/.ssh/authorized_keys を実行するなどして、空っぽのファイルとして用意する。
									③gitユーザー用の公開鍵ファイルを ~/（ユーザーID）.pub（例：~/git.pub）に格納する。
									内容例：（1行で記述）
									ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAvYFLofBoXrBst8t+Hg9a6PFWoQt2/HVp+0cTd6O9SBp44pH7C4rYSVyZFP/JXWwB9W5RJOgj1LxvktzKY9jeGWfaGrngEcLd/jGmI3x3alq90CStA2fbYIjsA/Sj6ncRwaXuoLUIrVNQka3gfJNlq5leVo695xf+hSsnTThgJYMxeMnZ0+tpuemQhqwx6uM1LXXzE6ExhsUgVXO0u+82/YrqdwwhMCAHe8AitC5e6ChqkHkMbh9oWNsvqlMas0OVO0oW7Z2y+ie1HXtjnJjwyfQEJXolgEZTkWpRGnvOAWhXqu/pDRaUBWGmVz7qcZXX2epU2azRrf5PSEL3GMiNcw== git
									④~/bin ディレクトリ作成。
									⑤~/bin ディレクトリをパスに通す。
									例：~/.bashrc に下記の行を追加。
									PATH=~/bin:$PATH
									※この後の操作でパスが通っている必要があるので、$ source ~/.bashrc を実行してパスを通すか、git ユーザーでログインし直す。
									⑥gitolite ソースの install スクリプトを実行。
									$ gitolite/install -to ~/bin
									⑦gitolite のセットアップコマンドを実行。
									$ gitolite setup -pk ~/git.pub
									以上でインストール完了。
									
									なお、上記SSHのケースでのセットアップ方法に、git-shellをgit専用ユーザーの専用シェルとして、/etc/passwd に指定する方法を示しているが、gitolite使用時はその設定を行ってはいけない。（gitoliteが正常に動作しなくなる。）
										◎	◎

パターンNo.	[1] サーバー構成名	[2] プラットフォーム（OS）	[3] 評価に使用したOS・アプリケーションのバージョン	[4] リポジトリアクセスの通信方式	[5] ユーザー管理	[6] リポジトリ認証方式	[7] リポジトリアクセス時のパスワード／パスフレーズ指定方法	[8] 対応VCS	[9] セットアップ	[10] 社外からのアクセス	[11] 通信の安全性※機密性の保護
10	**ALMinium**	Linux ※Windowsは未評価	CentOS6.4 Git1.7.1-3 ALMinium 2.2.4-3 ※ALMiniumの中に、Redmine+プラグイン、Apache、Jenkins、MySQL、Ruby、Railsが含まれている。	http/https/svn	独自ユーザー管理 ※ユーザー登録時のみActiveDirectory（LDAP）統合が可能 ※ActiveDirectoryを使用するには、管理者ユーザーでログインした後、[管理]→[LDAP認証]で認証方式の追加と設定を行う必要あり。 （設定例） 名称：Active Directory ホスト：server.domain.local ポート：389 LDAPS：off アカウント：DOMAIN¥Administrator パスワード：******** 検索範囲：DC=domain,DC=local LDAPフィルタ：（空欄） タイムアウト：（空欄） あわせてユーザーを作成：on ログイン：sAMAccountName 名前：givenName 苗字：sN メールアドレス：mail ※ユーザー登録後、Active Directory側でパスワードを変更しても反映されない。ユーザーの初期登録時のみに影響。	http(s)のBasic認証 ※内部的にはRedmineのユーザー認証が用いられる ※公開リポジトリは認証なしでpullできる。（pushするには認証必要。） ※公開リポジトリは、Webからもログインせずにか確認可能。（なお、Web画面からは、参加ユーザーなどの情報が丸見えになる。また、リポジトリのアクセスURLが表示されていない。） ※svn専用プロトコル通信時はSubversion使用時のみ（？）	上記WebDAVのケースと同じ。	Git Mercurial Bazaar Subversion CVS Darcs	非常に簡単。 ※ただし、一発のインストールで多数のアプリケーションがまとめてインストールされる。バックアップとリカバリは別途行う必要あり。問題発生時は大変になる可能性あり。	○	○ ※https使用時（httpは危険）
11	**GitLab** ※GitHubクローンの一つ。	Linux ※Windowsは未評価	CentOS6.4 Git1.7.1-3 GitLab 5.2-stable ※他、RubyやRails、MySQLなど、GitLabインストール要件に該当する多数のアプリケーションやライブラリ、開発ツールをインストール（巷のインストール解説Webページ参照）	ssh/http(s)	独自ユーザー管理＆ ActiveDirectory（LDAP）統合 ※ActiveDirectoryを使用するには、gitlab の config/gitlab.yml の ldap: セクションにLDAPの認証法規を設定する必要あり。 また、ActiveDirectoryユーザーにはメールアドレスの登録が必須。 平文のパスワードを記述する必要があるのが難点。 （設定例） ## LDAP settings ldap: enabled: true host: 'server.domain.local' base: 'DC=domain,DC=local' # port: 636 port: 389 uid: 'sAMAccountName' method: 'plain' # "ssl" or "plain" bind_dn: 'CN=Administrator,CN=Users,DC=domain,DC=local' ※LDAPユーザーと独自ユーザーは、ログイン画面でどちらのユーザーでログインするか指定する必要がある。	ssh方式の場合 … RSA（公開鍵認証） http(s)の場合 … Basic/Digest認証（どちらか不明） ※GitHub同様に、複数のアクセス方式を提供している。 ※公開リポジトリは認証なしでpullできる。（pushするには認証必要。） ※Web上には公開リポジトリの情報も表示されない。 （GitHubでは見れる。） ただし、URLさえわかればアクセスできる。	ssh方式の場合 … 上記SSHのケースと同じ。 http(s)の場合 … 上記WebDAVのケースと同じ。	Git	割と手間。 細かい手順は省略するが、下記のインストールが必要。 ・Ruby, Rails, MySQL, GitLab-Shell, GitLab, NGINX（Apacheでも良い） ※予めGitLab用のユーザー（gitなど）を作っておく。 ※設定ファイルを編集し、通信方式やladap通信、データベースアクセス設定などを調整する必要あり。 ※OS起動時の自動スタート設定では、GitLab のスタートが、NGINX（WEBサーバー）より先でないとうまくいかない。	◎ ※単純にインストールするとhttp通信になるので、設定ファイルconfig/gitlab.ymlとWebサーバー(nginxやapache)を適切に設定する事でhttps通信可能。（未検証）	◎ ※ssh使用時が一番安全 ※http使用時は危険
12	**Gitorious** ※GitHubクローンの一つ。 ※未検証（特にインストールが複雑そうだった事もあり、検証対象から完全に除	-	-	-	-	-	-	-	-	-	-
13	**RhodeCode（Linux）** ※GitHubクローンの一つ。 ※一番お勧めの構成。	Linux	CentOS6.4 Python 2.7.3 ※ソースコードからインストール Python Virtual Env ※必須ではないが、今回使用 SQLite3.6.20-1 Git1.8.3 ※ソースコードからインストール mercurial1.4-3 RhodeCode 1.7.1	http/https ※https対応は少し手間をかける必要あり（未検証だが可能なはず）	独自ユーザー管理＆ ActiveDirectory（LDAP）統合 ※ActiveDirectoryを使用するには、管理者ユーザーでログインした後、[Admin]→[LDAP]で認証方式の追加と設定を行う必要あり。 （設定例） Enable LDAP : on Host : server.domain.local Port : 389 Account : DOMAIN¥Administrator Password : ******** Connection security : No encryption Certificate Checks : NEVER Base DN : DC=domain,DC=local LDAP Filter : （空欄） LDAP Search Scope : SUBTREE Login Attribute : sAMAccountName First Name Attribute : givenName Last Name Attribute : sN E-mail Attribute : mail	http(s)のBasic/Digest認証（どちらか不明） ※https対応は少し手間をかける必要あり（未検証だが可能なはず） ※公開リポジトリは認証なしでpullできる。（pushするには認証必要。） ※公開リポジトリは、Webからもログインせずにか確認可能。	上記WebDAVのケースと同じ。	Git Mercurial	やや手間。（難しくはない） 基本的にはPythonをインストールすれば、Python の easy_install コマンド（Linux の yum コマンドや Ruby の gem コマンドのようなものの Python 版）を使用して、RhodeCode をインストールできる。 インストールの流れは下記の通り。（必要なソフトは環境により異なる可能性あり。gcc などの開発ツールは最低限必要。） ※RhodeCodeの利用を推奨する事もあり、詳しく説明。 **・必要な各種ソフトをインストール** $ su - # yum install python sqlite python-devel git hg curl **・git1.8以上をソースからインストール** # yum install curl-devel expat-devel openssl-devel gettext perl-ExtUtilities-MakeMaker tcl tk tcl-devel tk-devel # cd /usr/local/src # git clone https://github.com/git/git.git # cd git # git tag # git chekout v1.8.3.1 ※もっとも新しいタグを指定	○	△ ※https使用時（httpは危険）

		[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
パターン No.	サーバー構成名	プラットフォーム（OS）	評価に使用した OS・アプリケーションの バージョン	リポジトリ アクセスの 通信方式	ユーザー管理	リポジトリ認証方式	リポジトリアクセス時の パスワード／パスフレーズ指定方法	対応VCS	セットアップ	社外からのアクセス	通信の安全性 ※機密性の保護

									・PythonのVirtualEnvをインストール ※Vertual Envは、Pythonの実行環境を幾つも複製する場合に使用する
									# cd /usr/local/src
									# curl -O http://python-distribute.org/distribute_setup.py
									# easy_install virtual_env
									・rhodecode ユーザーを作成
									# adduser rhodecode
									・rhodecode 実行フォルダを準備
									# cd /var/lib/
									# mkdir rhodecode
									# cdown rhodecode:rhodecode rhodecode
									・rhodecode 実行用のVirtualEnv環境を構築
									# su - rhodecode
									$ cd /var/lib
									・VirtualEnvに切り替え
									$ cd rhodecode
									$ source ./bin/activate
									・リポジトリ用の先頭ディレクトリを予め作成
									$ mkdir repos
									・RhodeCode をインストール
									$ easy_install rhodecode
									・RhodeCodeセットアップ用のコンフィグを生成
									$ paster make-config RhodeCode production.ini
									・セットアップを実行
									$ paster setup-rhodecode production.ini
									※途中、データベース新規作成の許可、リポジトリの先頭パス、管理者ユーザーのユーザー名とパスワード、e-mailアドレスが聞かれる為、都度入力
									Are you sure to destroy old database ? [y/n] y
									Enter a valid absolute path to store repositories. All repositories in that path will be added automatically:
									/var/lib/rhodecode/repos/
									Specify admin username: admin
									Specify admin password (min 6 chars): ********
									Confirm password: ********
									Specify admin email: admin@domain.local
									・設定ファイルを調整
									$ vi production.ini
									67行目：host = hstname.domain.local
									162行目：default_encoding = utf8, cp932 ※UTF-8とシフトJISコードのファイル内容がWeb上で表示できるようになる
									・ファイアウォールでtcpのポート5000番を通過可能に
									$ su - ※スーパーユーザー
									# vi /etc/sysconfig/iptables
									※下記の行を「-A INPUT」群の最後に追加。
									-A INPUT -m state --state NEW -m tcp -p tcp --dport 5000 -j ACCEPT
									# service iptables restart
									・RhodeCode を起動
									# su - rhodecode
									・次回以降のRhodeCode 起動手順
									# su - rhodecode
									$ cd /var/lib/rhodecode
									$ source ./bin/activate
									$ paster serve production.ini
									・バックグラウンドで実行する方法
									$ paster serve production.ini --daemon
									・バックグラウンド実行を停止する方法
									$ cat /var/lib/rhodecode2/paster.pid
									31993
									$ kill -TERM 31993		

		[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
パターンNo.	サーバー構成名	プラットフォーム（OS）	評価に使用したOS・アプリケーションのバージョン	リポジトリアクセスの通信方式	ユーザー管理	リポジトリ認証方式	リポジトリアクセス時のパスワード／パスフレーズ指定方法	対応VCS	セットアップ	社外からのアクセス	通信の安全性※機密性の保護
									・OS起動／終了に伴う自動起動設定を行う 下記URLに示されている /etc/init.d/paste-serve-rhodecode をほぼそのまま流用 http://mbrownnyc.wordpress.com/technology-solutions/rhodecode-and-redmine/part-2-install-rhodecode/ $ su - # cd /etc/init.d # vi paste-server-rhodecode ※上記URL内のソースの内容をコピペ ※一部書き換え： /var/run/paste → /var/lib/rhodecode /var/www/rhodecode-venv → /var/lib/rhodecode /var/log → /var/lib/rhodecode --user=paste → 削除 su -c "/var/lib/rhodecode/bin/paster ..." paste → su -c "cd /var/lib/rhodecode;source bin/activate;bin/paster ..." -rhodecode viエディタでの一括置換コマンド（一部） :1,$s/¥/var¥/run¥/paste/¥/var¥/lib¥/rhodecode/g :1,$s/¥/var¥/www¥/rhodecode- # chmod +x paste-server-rhodecode # chkconfig --add paste-server-rhodecode # chkconfig paste-server-rhodecode on # service paste-server-start **・今後やった方がよい事** - データベースを SQLite から MySQL もしくは PostgreSQL に変更してホットバックアップ（データベースとリポジトリのバックアップさえあれば RhodeCode は復元できる為、信頼性のあるバックアップを行った方が良い。） - https通信対応。ApacheまたはNginxを表に出し、ポート443のパスからのリバースプロキシで接続する方法が良さそう。		
14	**RhodeCode(Windows)** **※未評価** **※Windowsファイルのバックアップが行われている環境であれば、この構成を実現するのが一番良い。確認していないが、対応の実績はあるらしい。**	Windows	-	（同上）	（同上）	（同上）	（同上）	（同上）	※ほぼ同上。ただし、扱われるパスや自動起動の方法は大きく異なる。設定ファイルの扱いなども変わるはず。 Windows版のインストールを解説するサイトでは、easy_installの他、pip の利用しているものが見受けられる。	（同上）	（同上）
15	**【参考】GitHub** **※ソーシャルホスティングサービス** **※その他、BitBucketやStashなどの有名なサービスが幾つかあるが、GitHubのみを取り上げる。** **※GitHub Enterpriseによる、独自GitHubサーバーの導入は未検証。**	-	-	https/ssh/Git専用プロトコル	GitHub独自ユーザー管理	・https通信時 … Digest認証 ・ssh通信時 … RSA（公開鍵認証） ・Git専用プロトコル通信時 … 認証なし（Pullのみ可）		Git	セットアップ不要。 既存の一般向けホスティングサービス（クラウドサービス）を利用する。 企業内に専用のGitHubサーバーを立てる為のエンタープライズライセンス（https://enterprise.github.com/）もある。 耐障害性や、他社との連携を考えた場合、自社サーバーを立てるエンタープライズよりも、既存のホスティングサービスを利用した方が低コストでよさそう。	○	◎ ※ssh使用時が一番安全

Gitサーバー比較

		[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
パターン No.	サーバー構成名	メンテナンス方法						日本語対応				
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI 有無	基本画面	ファイル名	コミットログ	ファイル内容表示	
											UTF-8	Shift-JIS
1	**Windowsファイル共有 (共有フォルダ)**	Windows／ActiveDirectoryのユーザー登録方法に従う。	共有フォルダに対するアクセス権限および共有の設定で制御。 ※Windowsの方がより細かい設定が可能。	Windows／ActiveDirectoryのユーザー管理方法に従う。	共有フォルダ上でTortoiseSVNからリポジトリを新規作成する。 作業ファイルを持たない Bare（裸）リポジトリを指定して生成する。	不要 ※ユーザーの数だけリポジトリのコピーがあるので、無理にバックアップを行う必要は無い。 障害発生時は別の場所に新しいリポジトリをすぐにコピーできる。 ※心配なら別のサーバーに定期的にコピーを作成するなども良い。	×	-	-	-	-	-
2	**Gitデーモン（サービス）** **※未評価**	-	なし。	-	Git管理者がプロジェクトからの依頼に応じてサーバー上で直接追加。 もしくは、sambaやNFSを使用して、リポジトリ用のフォルダをファイル共有で公開しておき、共有フォルダへのアクセスが許されたユーザーが直接 TortoiseGit でリポジトリを生成する。 いずれにせよ、git init --bare --share コマンドで初期化し、作業ファイルを持たない Bare（裸）リポジトリを生成する。	（同上）	×	-	-	-	-	-
3	**SSH** **※Git使用時に最も標準とされる通信方式**	公開鍵をサーバーに配置するための独自の仕組みが必要。 もしくは、専任の管理者がユーザーから公開鍵ファイルを受け取って手作業で配置する。 配置先は、サーバー内の /home/git/.ssh/authorized_keys ファイル。1行につき１ユーザーの公開鍵を記述する。 所定の共有フォルダに置かれたファイルを自動的に収集してauthorized_keys ファイルを生成する仕組みなどが簡単に作れて良い。 なお、鍵の生成はパスフレーズを入力する都合もある為、各ユーザー毎に自分で行ってもらう。TortoiseGit 付属の鍵生成ツール puttygenを使って生成する。秘密鍵はそのまま各自で管理してもらう。（詳しいマニュアルを用意する必要あり）	なし。	各ユーザーは、TortoiseGit付属の鍵生成ツール puttygenを使ってパスフレーズを変更できる。 既存の秘密鍵をツールにインポートして、パスフレーズだけ変えて秘密鍵を変更すれば、公開鍵は変更せずにそのまま使える。 ※公開鍵認証方式は、いかなる場面でも秘密鍵とパスフレーズを自分のPCの外に出す事がない為安心。通常のユーザー認証時に入力するパスフレーズも、ローカルで秘密鍵の復号に用いられるだけなので、ネットには流れない。	（同上） ※（注）ファイルのオーナー、アクセス権は、sshユーザーから読み書きできるように設定しておく。	（同上）	×	-	-	-	-	-

		[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
パターン No.	サーバー構成名	メンテナンス方法										
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI 有無	日本語対応				
								基本画面	ファイル名	コミットログ	ファイル内容表示	
											UTF-8	Shift-JIS
4	**WebDAV（Basic／Digest認証）** **※http(s)通信で認証なしの場合も同様** **※Subversionでは最も一般的な構成（特にWindowsプラットフォーム）**	Apache、IISなどのWebページの設定方法に従う。	なし。	Apache、IISなどのWebページの設定方法に従う。 管理者が一手に設定を担う。	（同上） ※（注）ファイルのオーナー、アクセス権は、apacheの実行ユーザーから読み書きできるように設定しておく。	（同上）	×	-	-	-	-	-
5	**WebDAV(NTLM認証)**	ActiveDirectoryのユーザー登録方法に従う。	なし	ActiveDirectoryのユーザー管理方法に従う。 Windowsのパスワード変更が反映される。	（同上）	（同上）	×	-	-	-	-	-

		[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
パターン No.	サーバー構成名	メンテナンス方法										
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI 有無	日本語対応				
								基本画面	ファイル名	コミットログ	ファイル内容表示	
											UTF-8	Shift-JIS
6	**WebDAV(SSPI認証)** **※検証不十分** **※Subversionでは一般的な構成の一つ**	（同上）	なし	（同上）	（同上）	（同上）	×	-	-	-	-	-
以下、管理ツールを導入する構成												
7	**gitweb** **※リポジトリの状態確認用の補助ツール。WebDAV,SSH,gitosis,gitliteと組み合わせて使用する事が可能。プラットフォームやアプリケーション、セットアップ方法等の説明については上記WebDAVの構成を参…**	-	-	-	-	-	○	英語	○	○	○	×
8	**gitosis** **※未評価（後継ツールとも言うべきgitoliteのみ検証し、gitosisについては一切の調査**	-	-	-	-	-	-	-	-	-	-	-
9	**gitolite**	管理用Gitリポジトリ（インストール時に自動生成される「gitolite-admin」）上の設定ファイル const/gitolite.conf を編集・コミットしてPushする。 （最初にリポジトリのcloneを作成してから編集する。） また、新しくユーザーを追加した場合、そのユーザーの公開鍵ファイルを、同管理用gitリポジトリの所定のフォルダに、（ユーザー名）.pub というファイル名で追加・コミットしてpushする。 ※公開鍵の生成については、上記SSH参照。 ※gitolite-admin リポジトリへのアクセス権限は管理者のみにして、公開鍵は管理者が受け取って対応した… ※ファイル構成例 [gitolite-admin] |-[conf] | `- gitolite.conf … ユーザーとグループ、リポジトリ、アクセス権限の設定 `-[keydir] |- user_a.pub … ユーザー毎の公開鍵 |- user_b.pub |- user_c.pub . . . ※gitolite.conf の内容例（詳しい説明は省略） @developer = user_a user_b … グループの設定 @all_staff = @developer user_a repo gitolite-admin …リポジトリとアクセス権の設定 RW+ = user_a - = @all repo proj1 proj2 … リポジトリとアクセス権の設定 RW+ = @all repo proj3/sbu1 proj3/sub2 … リポジトリとアクセス権の設定 RW+ = user_a R = @developer - = @all	管理用Gitリポジトリ「gitolite-admin」上の設定ファイル conf/gitolite.conf を編集・コミットしてPushする。 「repo 」で始まるリポジトリ指定行の直下に続けて、「アクセス権 = 対象グループ／ユーザー群」の表示を列挙する。 （左記参照） ※リポジトリ毎や、ブランチ毎にアクセス権を設定できる点が良い。 ※設定に間違いがあると、全てのリポジトリにアクセスできなくなったりといった事故が起こる為、厳重注…	上記SSHの説明参照。	管理用Gitリポジトリ「gitolite-admin」上の設定ファイル conf/gitolite.conf を編集・コミットしてPushする。 「repo 」で始まるリポジトリ指定行の右にリポジトリ名を列挙する。複数指定や、サブディレクトリを含んだ指定も可能。 （左記アクセス権の説明参照） ※設定に間違いがあると、全てのリポジトリにアクセスできなくなったりといった事故が起こる為、厳重注意。	（同上） ただし、管理用Gitリポジトリ「gitolite-admin」を正しく所定の場所に復旧する必要あり。	×	-	-	-	-	-

パターンNo.	サーバー構成名	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
		メンテナンス方法						日本語対応				
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI有無	基本画面	ファイル名	コミットログ	ファイル内容表示	
											UTF-8	Shift-JIS
10	**ALMinium**	管理者は、Web（Redmine）上でユーザやグループの登録が行える。 ※LDAPの設定を行った上で、ActiveDirectory（AD）のユーザーでログインすると、自動的にALMiumのユーザーが登録される為、最初にADにユーザーを登録しておく。ADに存在しないユーザとの併用も可能。 ※グループ設定はADのグループとは連動しない。 ※ユーザー登録を依頼する機能がログイン画面にある。ここで登録依頼されたユーザーは、管理者が有効化する事ができる。	管理者は、Web（Redmine）上でプロジェクトに対するユーザー／グループのアクセス権を設定できる。 リポジトリ操作以外にも、Redmine上の役割など、かなり細かい権限をまとめたロール（役割）を編集して扱うことができる。 ※基本的に、特定のプロジェクトに対する特定のグループのロール（役割）を変更する事で、読み取り専用に変えるなどの操作を行う。 ※同一プロジェクト内の複数のリポジトリに別々の権限を設定したり、ブランチ毎に権限を設定したりという事はできない。	ユーザー各自は、Web（Redmine）上から自分の表示名やパスワード、アバター画像などを変更できる。 ※Active Directory側でパスワードを変更しても反映されない。初回のユーザー登録時のみ反映される様子。	管理者は、Web（Redmine）上でプロジェクトを登録すると、そのプロジェクトの管理項目の一つとしてリポジトリを登録することができる。 リポジトリは複数、Git以外のもの（SubversionやMercurialなど）も含めて扱う事が可能。 Web上でリポジトリを登録すると、実際にBareリポジトリがサーバー上に生成される。 公開プロジェクトとして設定されたプロジェクトのリポジトリは、ALMiniumのユーザーでなくてもアクセスできる。 ※Redmineのプロジェクトとリポジトリが完全に連動しており、Redmineをしっかり使う場合以外では扱いにくい。	Gitリポジトリ、Redmineのデータ（MySQL）、Jenkinsの設定（および実行ログ）、Apacheの設定をそれぞれバックアップし、復旧できるようにする必要あり。 問題が発生した場合は面倒になる可能性あり。 仮想マシンで運用するなら、定期的にスナップショットを残し、問題発生前のスナップショットまでロールバックできるやり方が一番簡単。（当然その間の更新は失われる）	○	日本語	×	○	○	×
11	**GitLab** ※GitHubクローンの一つ。	管理者は、Web上でユーザやチーム（グループ）の登録が行える。 ※LDAPの設定を行った上で、ActiveDirectory（AD）のユーザーでログインすると、自動的にGitLabのユーザーが登録される為、最初にADにユーザーを登録しておく。ADに存在しないユーザとの併用も可能。 ※チーム設定はADのグループとは連動しない。 ※ユーザーのアイコン管理は、GitHubと同じく、Gravatar サービスを利用し、メールアドレスとアイコン画像を登録する。	管理者は、Web上でプロジェクトに対するユーザーのアクセス権を設定できる。チームのメンバーをまとめて登録することもできる。（チームに対する権限設定はない。） ユーザーに設定可能なアクセス権（役割）は、Guest, Reporter, Developer, Master の4つだけ。 リポジトリに対するアクセス権の他、そのリポジトリに対するIssueの発行など、機能的な制約が扱われる。 リポジトリグループを扱う事ができ、グループ内のリポジトリ全てのアクセス権を一括変更する事ができる。 （ただし、アクセス権の削除は一括では行えない。） ※ブランチやタグ毎に権限を設定したりという事はできない。	ユーザー各自は、Web上から自分の表示名やパスワードを変更できる。 LDAP設定している場合、ActiveDirectoryのユーザー管理方法に従う。 Windowsのパスワード変更が反映される。 また、リポジトリのアクセスにssh通信を用いる場合は、各ユーザー自身がWeb上の操作で公開鍵を登録する。 （My Profile 画面で、[Add Public Key]ボタンを押し、公開鍵のテキストをコピペする。タイトルは任意の文字列。）	ユーザーは、Web上でプロジェクトを登録すると、実際にサーバー上にBareリポジトリが作成される。（一つのプロジェクトは一つのリポジトリを持つ） リポジトリグループを作成して、幾つかのリポジトリをまとめる事ができる。 公開リポジトリとして設定されたリポジトリは、GitLabの登録ユーザーでなくても読み取る事ができる。 書き込みは、許可されたユーザーのみ。	Gitリポジトリ、GitLabのデータ（MySQL）、GitLabの設定、NGINX(Apache)の設定をそれぞれバックアップし、復旧できるようにする必要あり。 リポジトリとMySQLのバックアップさえあれば、おそらくきちんと復旧できる（未確認）。 MySQLは定期的なホットバックアップの仕組みを構築する事も必要。 仮想マシンで運用するなら、ALMiniumのケースと同じく、定期的にスナップショットを残し、問題発生前のスナップショットまでロールバックできるやり方が一番簡単。（当然その間の更新は失われる）	○	英語	○	○	○	×
12	**Gitorious** ※GitHubクローンの一つ。 **※未検証（特にインストールが複雑そうだった事もあり、検証対象から完全に除**	-	-	-	-	-	-	-	-	-	-	-
13	**RhodeCode（Linux）** **※GitHubクローンの一つ。** **※一番お勧めの構成。**	管理者は、Web上でユーザやユーザーグループの登録が行える。 ※LDAPの設定を行った上で、ActiveDirectory（AD）のユーザーでログインすると、自動的にRhodeCodeのユーザーが登録される為、最初にADにユーザーを登録しておく。ADに存在しないユーザとの併用も可能。 ※ユーザーグループ設定はADのグループとは連動しない。 ※ユーザーのアイコン管理は、GitHubと同じく、Gravatar サービスを利用し、メールアドレスとアイコン画像を登録する。	管理者は、Web上でプロジェクトに対するユーザーおよびユーザーグループのアクセス権を設定できる。 リポジトリ毎に、ユーザー個別、もしくは、グループに対するアクセス権を設定できる。 設定可能なアクセス権（役割）は、none（読み書き付加）、read（読み取り専用）、write（読み書き可）、admin（管理者） の4つだけ。 UIもわかり易く、権限設定もシンプルでとても扱い易い。 また、複数のリポジトリを束ねたリポジトリグループ（実質的なプロジェクト）も扱う事ができ、リポジトリグループに対する、ユーザーおよびユーザーグループのアクセス権を設定できる。 ※ブランチやタグ毎に権限を設定したりという事はできない。	ユーザー各自は、Web上から自分の表示名やパスワードを変更できる。 LDAP設定している場合、ActiveDirectoryのユーザー管理方法に従う。 Windowsのパスワード変更が反映される。	ユーザーは、Web上でリポジトリを登録すると、実際にサーバー上にBareリポジトリが作成される。 リポジトリグループを作成して、幾つかのリポジトリをまとめる事ができる。 公開リポジトリとして設定されたリポジトリは、RhodeCodeの登録ユーザーでなくてもアクセスできる。 読み取り専用／読み書き許可のどのようなアクセス権を与えるかは、リポジトリ毎に設定でき、公開リポジトリであっても、隠す事ができる。	実際のリポジトリファイルのバックアップの他、SQLiteを使用している場合、rhodecode.db ファイルさえバックアップしていれば、新規に環境を構築し直しても簡単に復元できる。 新しい環境にリポジトリファイルとrhodecode.dbをコピーするだけで済む。 ※設定ファイル production.ini の設定し直しは別途必要だが、ごくわずかで済むはず。バックアップから復元しても良い。 ※復元の際にRhodeCodeのバージョンアップを伴うと、データベース構造が変わる可能性がある為、うまくいかない可能性がある。 ※できればSQLiteではなく、MySQLかPostgreSQLを用いて、信頼性の高いホットバックアップ（DB稼働中の安全なバックアップ）を行う方が良い。これは、定期実行で何世代か分のデータベースデータをダンプ出力する仕組みの事。データベースのデータ更新中に問題が生じると、データベース全体が破損する可能性がある事と、バックアップ中に行われるデータ更新の影響を受けず、信頼性のある（中途半端に更新されたデータを含まない）データとしてバックアップする事ができる為。	○	英語	○	○	○	○ ※要設定変更（セットアップの説明参照）

	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
パターン No.	サーバー構成名	メンテナンス方法					日本語対応				
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI 有無	基本画面	ファイル名	コミットログ	ファイル内容表示 UTF-8	ファイル内容表示 Shift-JIS

		[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]
パターン No.	サーバー構成名	メンテナンス方法										
		ユーザー登録方法	リポジトリに対するアクセス権管理方法	パスワード／パスフレーズ変更方法	リポジトリ作成方法 ※Bareリポジトリの作成	バックアップと復旧方法	GUI 有無	日本語対応 基本画面	ファイル名	コミットログ	ファイル内容表示 UTF-8	ファイル内容表示 Shift-JIS
14	**RhodeCode(Windows)** **※未評価** **※Windowsファイルのバックアップが行われている環境であれば、この構成を実現するのが一番良い。確認していないが、対応の実績はあるらしい。**	(同上)	(同上)	(同上)	(同上)	(同上)	○	英語	○	○	○	○
15	**【参考】GitHub** **※ソーシャルホスティングサービス** **※その他、BitBucketやStashなどの有名なサービスが幾つかあるが、GitHubのみを取り上げる。** **※GitHub Enterpriseによる、独自GitHubサーバーの導入は未検証。**	ユーザー各自が登録が、新規ユーザー登録画面で登録する。	無料のサービスでは、すべてのリポジトリが公開になる。 有料のプランに加入にする事で、プライベートリポジトリを作成する事ができ、特定のユーザーにアクセス権限を設定できる。	ユーザー各自は、Web上から自分の表示名やパスワードを変更できる。 また、リポジトリのアクセスにssh通信を用いる場合は、各ユーザー自身がWeb上の操作で公開鍵を登録する。 （Account Settings 画面の[SSH Keys]ページで、[Add SSH key]ボタンを押し、公開鍵のテキストをコピペする。タイトルは任意の文字列。）	ユーザーは、Web上でリポジトリを登録すると、実際にサーバー上にBareリポジトリが作成される。 公開リポジトリとして設定されたリポジトリは、GitHubの登録ユーザーでなくてもアクセスできる。 書き込みは、許可されたユーザーのみ。	-	○	英語	○	○	○	×

Gitサーバー比較

		[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI		特記事項	総評
		ユーザー追加操作	リポジトリ追加操作		
1	**Windowsファイル共有 (共有フォルダ)**	-	-	URLに file:// というプレフィックスを指定すると、通信用に別プロセスが立ち上がるため、処理速度が遅くなるらしいので、共有ファイルは普通のファイルパスのみで指定した方が良さそう。	【△】 最も手軽に導入できる方法。 問題発生時のリカバリも、最新リポジトリのコピー（誰かのスタッフのPCから得る）と代替のサーバーが用意できれば簡単に済むので、保守性も高い。 ただし、社外への公開ができない事や、ユーザーアクセス権の管理ができないといった問題点がある。 ※以下のどの構成を採用する場合でも、問題発生時の復旧までの間や、局所的な都合（小さなグループ内や個人間の受け渡し時）により、この方法での共有が一時的に必要になる事が起こりえる。
2	**Gitデーモン（サービス）** **※未評価**	-	-	処理速度は全ての手法の中で最も早いとの事。 公式にも、セキュリティの観点などからあまり利用を勧めてない。 Gitは、基本的にSubversionのようなユーザーアクセス権限の機能を持っていない。（他のサーバー製品はGitの手前でユーザーアクセス制御を行っている。） なお、不特定多数に無差別にリポジトリを公開するような場合には有効な方法。	【×】 簡単に導入できる。 上記の構成と同じく、問題発生時のリカバリも、最新リポジトリのコピー（誰かのスタッフのPCから得る）と代替のサーバーが用意できれば簡単に済むので、保守性も高い。 ただし、それなりに管理者の手間がかかる方法となる。 ファイアウォールを通す必要もある。（TCPポート9418） 特記事項に示す通り、セキュリティ上の問題があり、実運用は難しい。 また、共有リポジトリが、多数のサーバーに分散するような管理形態になっている場合（例えば開発プロジェクト毎にに別のファイルサーバーにリポジトリを置いている）、Gitデーモンを一つのサーバーで運用するのは困難。その場合、Gitデーモンもポートを変えるなどして複数起ち上げる必要がある。かつてsvnserveでそのような管
3	**SSH** **※Git使用時に最も標準とされる通信方式**	-	-	最も基本的な使用方法。 管理ソフトを使わないのであれば（sshやApacheなどの通信ソフト以外のリポジトリやユーザーを管理するソフトを用いない場合）、この方法が最も基本的で、パフォーマンスと安全性の両面に優れた方法となる。 ただし、ユーザーアクセス権限などの機能が不足している。 この事は、Git自体が本来そのような機能を有していない事を意味する。	【○】 導入自体は簡単。 上記までの構成と同じく、問題発生時のリカバリも、最新リポジトリのコピー（誰かのスタッフのPCから得る）と代替のサーバーが用意できれば簡単に済むので、保守性も高い。 ユーザー各自に鍵を生成してもらうなど、少し面倒を強いるが、きちんとマニュアル化されていれば大きな問題ではない。 ユーザーアクセス権限や高度な機能を必要としないなら、この方法が構成として一番単純な上、通信も比較的高速な上、通信データの安全性も高くて良い。 ユーザー毎に秘密鍵を持つようにするかなど、ポリシーを考える必要あり。 同じ秘密鍵を複数ユーザー間で使い回す事は可能。その方法を採用した場合、一人一人のユーザーに対応してもらう作業が減る為、管理コストが下がる。ただし、秘密鍵の受け渡しが頻繁になると秘密鍵の漏洩の危険性を高める事になる為、セキュリティの低下を招く。 退職者や取引終了者の所有する鍵を無効化する必要性を考慮すると、面倒でもユー 新しい鍵を発行する手間が面倒で、ユーザー間で秘密鍵を受け渡す事があった場合、検出が難しい。共通鍵の重複をチェックするスクリプトを用意する事も一つの対策になるが、万全ではない。そういった事がないように、ユーザーに対するモラルの教育も重要

		[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI		特記事項	総評
		ユーザー追加操作	リポジトリ追加操作		
4	**WebDAV（Basic／Digest認証）** **※http(s)通信で認証なしの場合も同様** **※Subversionでは最も一般的な構成（特にWindowsプラットフォーム）**	-	-	Gitリポジトリを WebDAV で公開する場合は、フックスクリプト post-update を設定し、git update-server-info コマンドを実行する必要がある。これにより、リポジトリの info/refs ファイルが更新され、正しいリポジトリの状態が扱われるようになる。 ※WebDAVを使用する手法としては、「認証なし」のスタイルで構築するケースも考えられる。ただし、セキュリティ上の危険が高い為、今回の比較対象からには含まないものとした。 ※別途gitwebというツールをインストールする事で、Web上でリポジトリの状態を確認する事が可能（今回の調査では未検証） ※【注意】http(s)通信時は、TortoiseGitでリポジトリのcloneを行う際、「Putty認証キーのロード」欄のチェックを外し、Putty認証キーが指定されていない状態にしておく。 （無駄にパスフレーズの入力を促される事がある。）	【△】 導入自体は簡単。 上記SSHの構成よりも手間がかかるが、さほど難しくはない。 上記までの構成と同じく、問題発生時のリカバリも、最新リポジトリのコピー（誰かのスタッフのPCから得る）と代替のサーバーが用意できれば簡単に済むので、保守性も高い。 ただし、セキュリティ設定（ユーザー認証）に際して、逐一管理者の手間がかかる。 リポジトリのURLに対して単一のパスワードを全ユーザーが共通利用する形態とするならば有効な手段。管理コストも低い。 ユーザーアクセス権の管理ができない点はやはりマイナス要因だが、わりきってそれを行わないのも手。 社外からのアクセスは、https通信のみを公開すべき。（http通信は公開しない。） 社内向けと社外向けの両方のインターフェースを備えた構成が扱い易い。 また、上記SSHの構成と組み合わせて、社外アクセスはSSHにするのも良い。 NTLM/SSPI認証との混在環境も構築可能。その場合、社外アクセスにだけDigest認証を適用するといった方法が考えられる。 Basic認証は仮に採用するとしても、社内アクセスにとどめるか、https必須とするべき。ただ、msysGitもDigest認証に対応しているので、Basic認証は無理に使う必要がない。
5	**WebDAV(NTLM認証)**	-	-	上記Basic/Digest認証と同じく、フックスクリプト post-update を設定し、git update-server-info コマンドを実行する必要あり。	【○】 導入はさほど難しくない。 上記のBasic/Digest認証よりも初期構築の手間がかかる。（WinBindを稼働させるなどの手間が必要。） ActiveDirectory統合認証は、管理者の手間がかからず、管理コスト面では Basic/Digest認証よりかなり良く、各ユーザーも自身のWindows(ActiveDirectory)ユーザーIDとパスワードを知っていれば良いので敷居が低い。 ただし、社外から接続するユーザーもActiveDirectoryにユーザー登録をする必要がある。（それを避けて、Digest認証やSSHと組み合わせる構成も考えられる。） 他は上記Basic/Digest認証のケースと同様。 特に、社外アクセスの方法はよく考慮すべき。 Subversionの管理がこの形態（もしくは下記のSSPI認証を用いた形態）を取るのが一般的である為、これまで稼働しているSubversion用のサーバーがあるなら、それと一体化するのは管理コスト面で有効。 なお、Subversionのように、リポジトリへのアクセス権限を管理する仕組みがない為、それを許容するかどうかが判断の大きなポイント。

		[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI		特記事項	総評
		ユーザー追加操作	リポジトリ追加操作		
6	**WebDAV(SSP1認証)** **※検証不十分** **※Subversionでは一般的な構成の一つ**	-	-	上記Basic/Digest認証と同じく、フックスクリプト post-update を設定し、git update-server-info コマンドを実行する必要あり。 【問題】https通信時にセッションが確立できずエラーとなる。OpenSSLのサーバー／クライアント間のバージョンの相性の問題か？原因不明。 error:14077458:SSL routines:SSL23_GET_SERVER_HELLO:reason(1112)	【×】 ※検証不十分につき「×」評価。下記「検証不十分の問題」を解消できれば【○】評価。 （同上） ※初期導入については、LinuxのNTLM認証のようにWinBindを稼働させる必要が無い為、NTLM認証よりは少しだけ手間が軽減される。 Windowsファイルサーバーのバックアップが標準化された環境にサーバーを立てるなら、Linuxを採用するよりもWindowsサーバーを稼働させる方が良い。 ただし、下記の問題を解消する必要がある。 【検証不十分の問題】なお、このWindowsでは、https通信がTortoiseGit(msysGit)との間でエラーが発生する（右記参照）。導入したApacheに付属のmod_sslのバ
以下、管理ツールを導入する構成					
7	**gitweb** **※リポジトリの状態確認用の補助ツール。WebDAV,SSH,gitosis,gitliteと組み合わせて使用する事が可能。プラットフォームやアプリケーション、セットアップ方法等の説明については上記WebDAVの構成を参照。**	-	-	-	-
8	**gitosis** **※未評価（後継ツールとも言うべきgitoliteのみ検証し、gitosisについては一切の調査**	-	-	-	-
9	**gitolite**	-	-	gitoliteは、sshアクセスは単一のLinuxユーザーのみを用いて、Git本来の管理とは異なるgitolite専用のユーザーを管理し、フックスクリプトの処理でアクセス権限を判断する仕組みとなっている。 なお、鍵に対してユーザーが割り当てられる仕組み。 管理用Gitリポジトリ「gitolite-admin」を、各プロジェクトの管理者が共同で編集する事になる為、競合に注意。 ※誤った設定を行うと、一切のリポジトリにアクセスできなくなる恐れがある。 　そのような場合、gitolite の管理から離れた形で、例えば、サーバーにログインして直接gitコマンドでcloneして編集、コミット、pushするなど。 ※【注意】pageant を使用して秘密鍵とパスフレーズを管理する場合、同一PC上で複数ユーザーを切り替えて使っている人は要注意。 gitoliteでは、単一のLinuxユーザーのみを使用するが、そのユーザーに全ユーザーの共通鍵が登録される。 pageantでは、自分が操作しようとしているリポジトリに関わらず（そのリポジトリに対してTortoiseGitで指定した鍵ファイルに関わらず）、サーバーに置かれている共通鍵に該当する秘密鍵が見つかると、優先的に選択されてしまう為、ユーザーを切り替える際は、pageantを終了させるか、秘密鍵を削除するなどする	【○】 非常に使い勝手が良い。 大きな利点として、リポジトリやブランチごとのユーザーアクセス権限を設定でき、グループを利用した迅速な設定変更も可能。 大事な局面で、一時的に読み取り専用に変えるといった対応も簡単にできる。 ブランチ毎のアクセス権設定は大きな強みの一つ。 サーバーに直接ログインしなくてもリポジトリを追加できる点は管理の敷居を下げて、各プロジェクトごとにリポジトリの管理者を用意するといった融通の利いた組織体制が期待できる。 ※ただし、編集ミスには非常に要注意。他のプロジェクトにも迷惑をかけることになる。 sshを利用する分、鍵ファイルの扱いなど、若干面倒が伴う。（この点については、Http/httpsによるActiveDirectory統合認証の方が管理コストは低い。ただし、sshの方が処理が早く、通信データ保全の安全性も高い。）

		[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI		特記事項	総評
		ユーザー追加操作	リポジトリ追加操作		
10	**ALMinium**	○	○	GUIは基本的にRedmine。Redmineを中心に、GitやMercurial、Subversionなどのリポジトリや、Jenkinsが統合された環境として利用できる。 かなりの高機能で、Wikiやニュース、文書の管理、タスク（チケット）管理、カレンダー、ガントチャート、アジャイルのバックログ、コードレビュー、Jenkins（Hudson）連動などの機能を備える。	【△】 リポジトリの管理機能だけで見ると、プロジェクト単位だけでリポジトリごとには権限が設定できなかったり、ブランチの権限を扱ったりという事ができない為、やや機能不足。 基本的には、Redmineを主体としたツールである為、かなりの高機能。 リポジトリのコミット（Push）とチケットの関連付けなどが容易に行える点がとても良い。 インストールが単純だが、管理者はMySQLのホットバックアップを別途設定するなど、全容を理解し、メンテナンスする高い知識が要求される。 設定系のUIはやや散漫な印象で、例えばリポジトリのアクセスパスを確認したい場合、どの画面に遷移すればいいのか直感的にわからない。（直感的に扱おうとすると失敗する） やや煩雑な画面構成。 Redmineなどのツールを試験的に使ってみるといったケースでは、非常に最適。
11	**GitLab** **※GitHubクローンの一つ。**	○	○	GitHubの機能・管理形態に最も近く、Issue（作業タスクのチケット）、Merge Request（リーダーにブランチのマージ依頼（GitHubのPull Requestと同様））、Wiki、Wall（ツイッターのようなもの）、Snippet（HitHubのGistのような直接ソースコードを書いて議論などに用いる）といった機能がある。	【○】 チーム（ユーザーグループ）に対するアクセス権の管理がないなど、同様のメンバーで幾つもの社内プロジェクトを扱う企業開発の考慮が若干足りていない印象。 やはりGitHubのようなソーシャル開発向け。 Web上のUIはシンプルでわかりやすい。 GitHubのように、http通信とssh通信を利用できる点はとても良い。 Http通信とWebのユーザー管理にはActiveDirectoryと連動できる為、扱いやすい。 おそらくGitHubクローンの中では最もシェアがある。 Web上の画面レイアウトも洗練されていてわかり易いが全て英語の為、Webを用いたIssueのやり取りなどをワークフローに組み込むのはやや敷居が高い。 とはえい、Merge Requestを用いて、安易なコードのアップが内容に防ぐワークフローはできるだけ実践した方がよい。 なお、総合的に、企業向けにはRhodeCodeの方が向いている印象。 （RhodeCodeはssh通信に対応していないのが若干残念ではある。）
12	**Gitorious** **※GitHubクローンの一つ。** **※未検証（特にインストールが複雑そうだった事もあり、検証対象から完全に除**	-	-	-	-
13	**RhodeCode（Linux）** **※GitHubクローンの一つ。** **※一番お勧めの構成。**	○	○	GitHubの機能・管理形態に近く、Fork（他者のリポジトリの作業用の複製）、Pull Request（リーダーにブランチのマージ依頼）といった機能がある。 ただし、Pull Request の機能は、現時点ではリポジトリにMercurialを指定した時だけしか使えない為、Gitでは活用できない。 また、リポジトリをわざわざ作るまでもないような、ちょっとしたソースコードやスクリプトを扱う為のGistや、特定のリポジトリの動向を監視する為のFollow（GitHubのWatchと同様）、Feedといった機能も備える。 その他、他の管理ツールに見られるような、Issue（作業タスクのチケット）の機能が無い。 なお、今回評価したバージョンでは、Webページのクライアントが IEの場合、互換モードにしないと正常に動作しなかった。 	【○】 企業での利用を考えると、ユーザー管理の仕組みが分かり易く、リポジトリの管理もシンプルで扱い易く、今回評価した中では最も導入し易いツールであった。 リポジトリの管理に特化したようなツールであり、ワークフロー上の管理ツールへの依存度は控え目で、問題発生時のリカバリは迅速に行えるものと考える。 日本語ファイルの対応も、他のツールよりも簡単に対応できて扱い易い。 Pull Request や Issue といった機能が使えないのが残念だが、RedmineやTracなどの他のツールと併用すれば、十分に補える。 http通信のままでは社外との通信に心配があるので、この点はうまく構成を変えるなどして、ApacheやNginxからのリバースプロキシーを用いて https 通信できる状態にすると良さそう。

	[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI	特記事項	総評
		ユーザー追加操作 / リポジトリ追加操作		

		[22]	[23]	[24]	[25]
パターン No.	サーバー構成名	GUI		特記事項	総評
		ユーザー追加操作	リポジトリ追加操作		
14	**RhodeCode(Windows)** **※未評価** **※Windowsファイルのバックアップが行われている環境であれば、この構成を実現するのが一番良い。確認していないが、対応の実績はあるらしい。**	○	○	（同上）	**【○】** 基本的に同上。 ただし、Windowsファイルのバックアップ環境を持つ場合は、LinuxよりもWindowsのサーバーとして立てた方が良い。
15	**【参考】GitHub** **※ソーシャルホスティングサービス** **※その他、BitBucketやStashなどの有名なサービスが幾つかあるが、GitHubのみを取り上げる。** **※GitHub Enterpriseによる、独自GitHubサーバーの導入は未検証。**	○	○	プライベートリポジトリを扱うには有料プランに加入する必要あり。 他、Gitのホスティングサービスの BitBucket、gitBREAKなどはプライベートリポジトリも無料で作成可能との事。 GitHubには企業向けのライセンスプラン（https://github.com/plans）もある。 また、GitHubにはソーシャル開発を助ける、Fork（他者のリポジトリの作業用の複製）、Pull Request（リーダーにブランチのマージ依頼）、Issue（作業タスクのチケット）、Wikiなどがある。 更に、各ユーザーやリポジトリの活動分析を行い、グラフやカレンダーに表示する機能なども備える。 また、リポジトリをわざわざ作るまでもないような、ちょっとしたソースコードやスクリプトを扱う為の	【△】 企業向けライセンスプランを適用するなどしてプライベートリポジトリを用いる事により、ホスティングサービスを企業内のクローズドな開発に利用し、提携会社間や拠点間で利用する方法も十分考えられる。 ただし、情報漏えいなどのセキュリティの心配や、サーバー障害やネットワーク障害で突然アクセスできなくなるアクシデントがないとは言い切れない。 ただし、その心配は自社サーバーを運営する場合も同じ事。懸念は社外に秘匿性の高いリポジトリを置く事をよしとできるかどうかに尽きる。