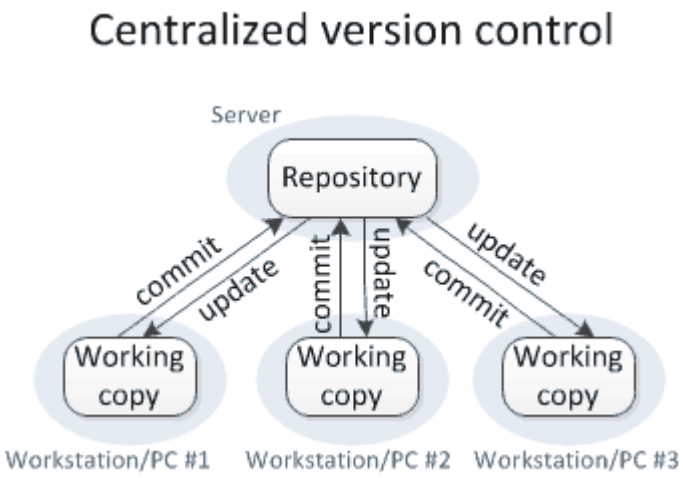
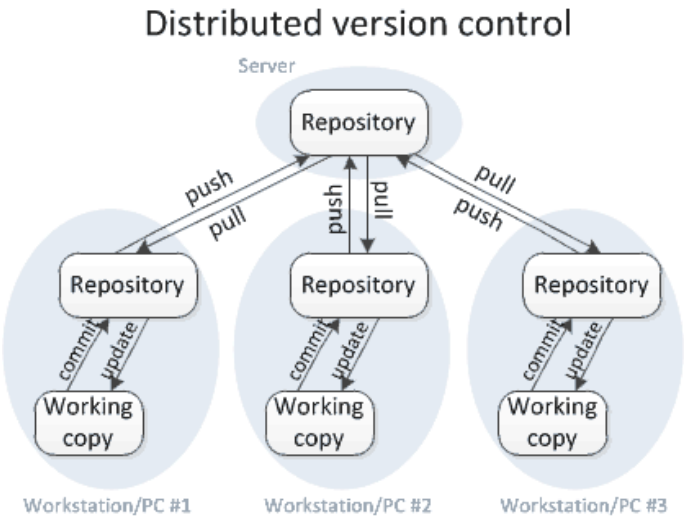
Version Control System (VCS)

VCS adalah salah satu bagian dari *Software Configuration Management* (SCM). Fokus utama pada VCS adalah menelusuri perubahan yang terjadi ketika kode sumber diubah dari satu versi ke versi lainnya (Brown, 2002). VCS digunakan untuk pengembangan perangkat lunak secara kolaboratif (dikerjakan oleh beberapa orang dalam satu tim atau dikerjakan oleh beberapa tim) karena setiap kode sumber yang baru dan diubah akan terekam perubahannya sehingga orang lain dapat mempelajari kode sumber melalui rekaman perubahan tadi. VCS sangat penting karena digunakan pada semua fase dari perencanaan dan pembangunan perangkat lunak (Costa dan Murta, 2013). Kegunaan dari VCS adalah untuk dengan mudah membedakan perubahan versi, pengambilan dan penelusuran, dan untuk menunjukkan hubungan antara berbagai versi (Ren dkk, 2010).

Menurut Stefan Otte (2009), terdapat dua model VCS yaitu *Centralized* VCS (CVCS) dan *Distributed* VCS (DVCS). Pada CVCS, terdapat satu *repository* yang secara garis besar adalah sebuah *server* yang menyimpan semua berkas termasuk sejarahnya. *Server* ini dapat diakses melalui jaringan lokal atau global. Setiap tahap sejarah dari berkas disebut sebagai *revision* atau *version* (biasanya berbentuk *integer*). Suatu *revision* terdiri dari berkas dan beberapa *metadata*. *Metadata* yang disimpan dapat berbeda tergantung dari *tools* yang digunakan. *Revision* yang paling baru disebut *head*. Pengembang dapat melakukan *checkout* pada suatu berkas pada *repository* dan dapat mengambil *revision* yang dia mau dari *repository*. Dengan melakukan *checkout*, pengembang mendapatkan salinan berkas pada komputernya. Salinan ini tidak mengandung sejarah tetapi pengembang dapat melakukan perubahan pada berkas tersebut. Setelah mengubah berkas, pengguna akan melakukan *commit* berkas tersebut ke *repository*. Pengembang juga menambahkan pesan tertentu ketika melakukan *commit* yang mana adalah deskripsi dari perubahan yang dilakukan pengembang. Pesan tersebut adalah bagian dari *metadata* yang akan disimpan sebagai sejarah dari berkas tersebut. Dengan melakukan *commit*, maka *revision* baru akan terbentuk dan nomor dari *revision* akan naik.



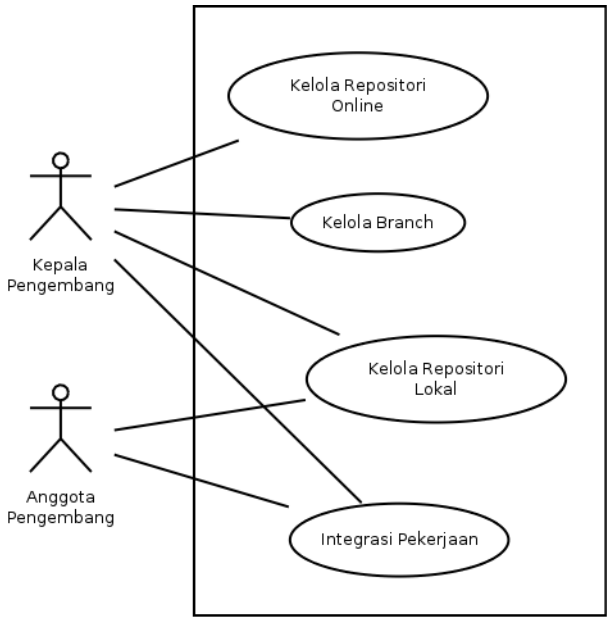
Pada DVCS, proses yang dilakukan pengembang terhadap berkas dan *repository* hampir sama. Perbedaannya adalah pada DVCS tidak memerlukan *repository server* pusat. Setiap pengembang memiliki *repository* pada komputer lokalnya yang disebut *local repository*. *Local repository* memungkinkan pengembang untuk bekerja secara *offline*. Kebanyakan operasi pada DVCS lebih cepat karena diakses secara lokal. Sebagai contoh, untuk memeriksa perubahan antara dua berkas hanya perlu operasi sederhana. Pada DVCS kegiatan *commit*, *branch*, *tag*, dan *checkout* sama dengan CVCS hanya saja pada DVCS *versioning* dilakukan pada *local repository*. Tetapi bagaimanapun pengembang harus membagi berkas ke pengembang lain sehingga pengembang menjadikan *repository* lokalnya bisa diakses melalui jaringan atau membuat *remote repository* pada *server* yang identik dengan *local repository*. Nantinya pengembang lain dapat melakukan *clone* pada *remote repository* untuk dimasukan ke *repository* lokalnya. Selain itu, model CVCS memiliki keterbatasan pada pengembangan yang dilakukan oleh tim secara bersama-sama (Alwis dan Sillito, 2009).



Menurut Carl Fredrik Malmsten (2010), pada DVCS, setiap pengembang memiliki *branch* masing-masing dari proyek dan pengembang lain tidak dapat melihat yang ada pada *branch* pengembang lain (ibarat salinan lokal dari proyek tersebut). *Branch* adalah sekumpulan dari versi berkas sumber yang berkembang, diidentikikasi sebagai *tag* yang seringkali mengidentifikasi versi berkas yang telah dan akan dirilis (IEEE, 2005). Pengembang bekerja tidak terpaut lokasi, artinya pengembang bisa mengerjakan dimana saja. Jika pengembang ingin melihat apa yang dikerjakan oleh pengembang lain adalah dengan meminta salinan pekerjaan pengambang lain itu sehingga bisa disatukan dengan pekerjaan yang dia lakukan. Selain untuk setiap pengembang, *branch* juga dibuat untuk setiap versi. Versi ini bisa versi rilis atau versi *build* dari aplikasi. Tujuannya adalah untuk membedakan antara versi satu dan lainnya dan merekan perubahan pada setiap evolusi kode sumber.

Aktifitas *Versioning* Secara Umum

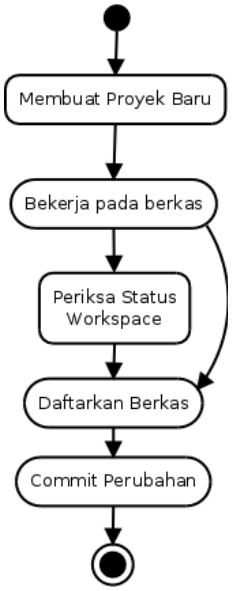
Aktivitas utama (*git-tower.com*, 2016) yang dilakukan oleh pengembang dalam kegiatan *versioning* adalah aktivitas dasar, pengelolaan fitur, dan kerja kolaborasi.



Aktifitas Dasar

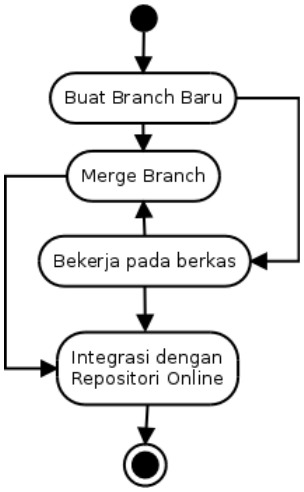
Aktifitas dasar adalah aktivitas utama dalam kegiatan *versioning* yang meliputi pembuatan repositori, pendaftaran perubahan, *commit* perubahan, dan status perubahan. *Workflow* dimulai dari membentuk repositori lokal, baik dengan membuat proyek baru atau meneruskan proyek lama.

Repositori lokal bisa dibentuk melalui *tools* VCS atau dengan menyalin repositori dari *Online Repository*. Ketika repositori sudah terbentuk, pengembang memeriksa status repositori seperti status *workspace*, status perubahan terakhir, dan status perubahan yang baru dilakukan. Selanjutnya pengembang mendaftarkan berkas-berkas yang akan direkam perubahannya yang nantinya akan menjari *log versioning*. Berkas-berkas yang didaftarkan di-*commit* agar sejarah perubahan terekam pada *log*. Hasil rekam perubahan ini bisa dilihat sebagai sejarah perubahan yang dihasilkan dari setiap *commit* yang dilakukan oleh pengembang. Dari sejarah ini, dapat dilihat kontribusi, produktivitas, dan peran pengembang dalam suatu proyek perangkat lunak. Hal ini dikarenakan data yang direkam tidak hanya perubahan tetapi juga status perubahan seperti waktu *commit*, author, *commit message*, dan status *merge*.



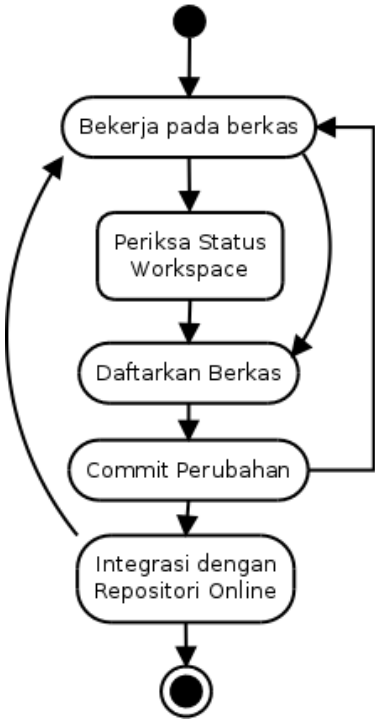
Pengelolaan Fitur

Pengelolaan fitur adalah kegiatan pengembang dalam pengerjaan fitur-fitur yang diterapkan pada perangkat lunak. Setiap fitur yang dikembangkan disimpan dalam *branch* terpisah agar modularitas perangkat lunak tetap terjaga. Nantinya *branch*-*branch* fitur ini akan diintegrasikan dengan melakukan *merge* terhadap beberapa *branch* sehingga pada akhirnya semua fitur dijadikan satu. Pengembang hanya dapat bekerja pada satu *branch* dalam satu waktu sehingga pengembang harus berpindah *branch* terlebih dahulu untuk mengembangkan fitur lain.



Kerja Kolaborasi

Kerja kolaborasi adalah kegiatan pengembang dalam mengelola suatu repositori bersama-sama dengan pengembang lain. Dalam berkolaborasi, pengembang memanfaatkan *Online Repository* dan mendistribusikan repositori lokal ke *Online Repository* dan pengembang lain sehingga repositori di setiap pengembang akan sama persis. Kesamaan tidak hanya pada sejarah perubahan, tetapi juga pada *branch* yang menyimpan fitur-fitur perangkat lunak. Pengembang memantau perubahan apa saja yang terjadi sejak terakhir kali integrasi perubahan melalui sinkronisasi repositori lokal dengan *Online Repository*. Integrasi perubahan akan membuat data di repositori lokal, baik sejarah maupun kode sumber, akan sama dengan yang ada di *Online Repository*. Pengembang melakukan sinkronisasi perubahan dengan *Online Repository* sehingga sejarah perubahan dapat disinkronisasi oleh pengembang lain. Pengembang membagikan fitur yang dibuatnya dengan pengembang lain melalui sinkronisasi ini. Diagram untuk kerja kolaborasi digambarkan pada gambar 2.6.



Perhatian Utama *Software Maintenance*

Xiaomin Wu dkk (2004) menyebutkan perhatian utama *software maintenance* berdasarkan pada 5W1H yaitu :

1. Apa yang terjadi sejak terakhir kali pengembang bekerja pada proyek?
2. Siapa yang melakukan perubahan?
3. Dimana hasil perubahan disimpan (seperti lokasi berkas, lokasi penambahan dan penghapusan, dan lain-lain)?
4. Kapan kegiatan perubahan dilakukan?
5. Mengapa dilakukan perubahan?
6. Bagaimana perubahan dilakukan (detil dari setiap perubahan)?

Perhatian utama *software maintenance* yang didefinisikan oleh Wu dkk (2004) berhubungan dengan kegiatan *versioning* yang dilakukan oleh pengembang. Kegiatan *versioning* merekam detil perubahan yang dilakukan oleh pengembang terkait berapa perubahan yang dilakukan, kapan perubahan dilakukan, siapa yang mengubah, tujuan perubahan, apa saja yang diubah, dan dimana perubahan disimpan. Hal-hal tersebut dibahas dalam perhatian utama *software maintenance.*

Best Practices Software Versioning

Versioning dilakukan oleh pengembang terhadap proyek perangkat lunak yang bermacam-macam sehingga menimbulkan perbedaan pada masing-masing pengembang dalam melakukan *versioning*. Dengan gaya yang berbeda ini, dirangkum beberapa *best practices* (washington.edu, Semver, dan Microsoft) yang bisa menjadi landasan pengembang dalam melakukan *versioning* :

1. Kelola repositori dengan baik. Repositori adalah ruang kerja yang digunakan oleh pengembang sehingga perlu diatur sedemikian rupa agar bisa dimanfaatkan dengan baik. Selain itu, repositori juga akan di-*share* dengan pengembang lain sehingga perlu diatur agar bisa terintegrasi dengan baik.

2. Gunakan pesan *commit* yang jelas dan deskriptif. Pesan *commit* adalah informasi yang diberikan pengembang mengenai perubahan yang dilakukannya terhadap kode sumber sehingga pesan *commit* harus bisa dimengerti oleh pengembang lain.

3. Selalu pilah berkas yang akan di-*commit*. VCS *tools* seperti Git dan Mercurial sudah bisa memilah secara otomatis sehingga sejarah yang akan direkam adalah berkas yang berubah statusnya. Namun pengembang pun harus tetap memilah berkas seperti berkas hasil eksekusi atau konfigurasi IDE tidak perlu di-*commit*.

4. Lakukan *commit* secara teratur. Setiap perubahan yang direkam harus bersifat *atomic* sehingga dapat dibedakan antara perubahan yang satu dengan yang lain. Dengan melakukan *commit* secara teratur pada setiap perubahan yang berarti, maka perubahan yang terjadi di setiap *commit* menjadi jelas. Selain itu, *commit* seara teratur juga membantu kerja kolaborasi dengan pengembang lain untuk menghindari konflik.

5. Hanya *commit* pekerjaan yang sudah selesai dan sudah diuji. Dalam konteks kerja kolaborasi, semua hasil pekerjaan dari masing-masing pengembang akan disatukan dan diuji. Melakukan *commit* terhadap pekerjaan yang belum selesai dan belum diuji akan menyebabkan kemungkinan terjadinya kesalahan saat integrasi pekerjaan semakin besar.

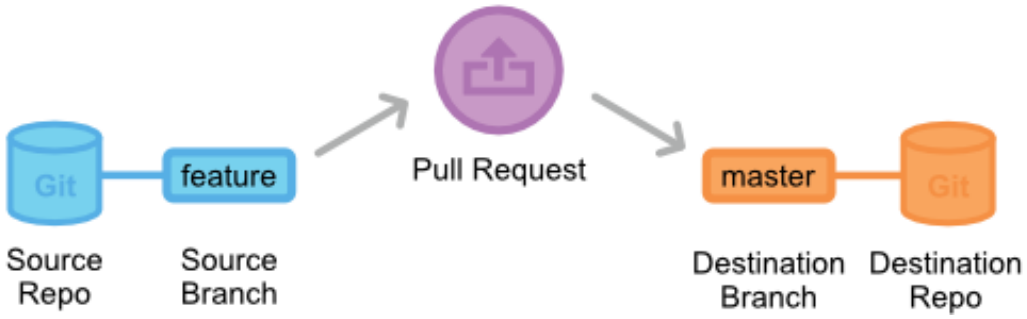
6. Jangan jadikan VCS sebagai *backup* terhadap pekerjaan. Jika ini dilakukan, akan sulit memilah mana *commit* hasil pekerjaan dengan *commit* penyimpanan sehingga sulit melakukan pengelolaan pekerjaan.

7. Patuhi *workflow*. Dengan perbedaan gaya pada masing-masing pengembang dalam melakukan *versioning*, perlu diterapkan suatu alur standar agar kegiatan *versioning* dalam kerja kolaborasi bisa konsisten.

8. Gunakan *branch*. *Branch* dapat dimanfaatkan untuk berbagai keperluan seperti memilah fitur, memisahkan antara pekerjaan yang selesai dengan yang masih dikembangkan, dan sebagai *workspace* masing-masing pengembang sehingga penggunaan dan pengelolaan *branch* menjadi penting.

DVCS *Workflow*

DVCS adalah *Distributed Version Control System* yaitu suatu model *versioning* yang mendistribusikan repositori tidak hanya di repositori pusat tetapi juga pada repositori lokal pengembang. DVCS memungkinkan konten pada repositori pusat dengan repositori lokal milik pengembang sama. DVCS *Workflow* adalah skema aliran kerja pengembang terhadap *branch codeline* pada git ketika melakukan *versioning* yang dapat diterapkan pada model DVCS. Terdapat 5 *workflow* pada Git *Workflow* (Atlassian) yaitu *Centralized Workflow, Feature Branch Workflow, Gitflow Workflow, Forking Workflow,* dan *Pull Request. Pull Request* adalah *workflow* yang paling banyak digunakan (diadaptasi pada GitHub, BitBucket, dan beberapa *versioning tools*) dan mendukung model DVCS karena pada *workflow* ini terdapat beberapa *branch* dengan *branch* master sebagai *branch* utama. *Branch* master dikelola oleh kepala pengembang dimana setiap pengembang memiliki *branch* masing-masing untuk pengembangan sendiri. Setiap pengembang menyelesaikan pengembangan, maka pengembang akan meminta izin penyatuan *branch* dengan *branch* master (dinamakan *pull* request). *Project manager* harus menyetujui terlebih dahulu agar *branch* bisa disatukan. Selain *branch* pengembang dan *branch* master, bisa juga dibuat *branch* lain seperti *branch test*ing atau *branch* versi untuk mempermudah pengembangan.



Eksplorasi *Versioning Tools*

Untuk mengetahui bagaimana cara kerja VCS, dilakukan eksplorasi terhadap VCS *tools* yang digunakan oleh pengembang dalam melakukan kegiatan *versioning*.

Overview

Ada banyak *versioning tools* yang digunakan pengembang dalam pengembangan perangkat lunak. Masing-masing *tools* menerapkan berbagai macam teknik dan alur yang dimaksudkan untuk mempermudah kerja pengembang. Namun menurut g2crowd.com dan zeroturnaround.com, Git dan Mercurial adalah *versioning tools* yang memiliki peringkat paling tinggi, baik dalam segi kepuasan maupun pengguna. Untuk mengetahui lebih dalam mengenai *versioning*, dilakukan eksplorasi terhadap dua *versioning tools* yaitu :

1. Git adalah *versioning control system* (VCS) yang banyak digunakan oleh *source-code management* (SCM) seperti GitHub dan BitBucket. Proyek aplikasi yang memanfaatkan Git salah satunya adalah Kernel Linux.
2. Mercurial (atau disingkat Hg) adalah VCS yang banyak digunakan oleh perusahaan dan organisasi pengembang aplikasi seperti Facebook, W3C, dan Mozilla.

Ketika melakukan ekplorasi, didapat beberapa perbedaan mendasar antara Git dengan Mercurial, yaitu :

1. Metoda kompresi. Pada perekaman metadata, Git menggunakan metoda *Zlib* sedangkan Mercurial menggunakan metoda *Delta Storage*.
2. *Branch*. Git memperlakukan *branch* sebagai subrepositori sehingga metadata antar *branch* terpisah satu sama lain. Sedangkan Mercurial memperlakukan *branch* sebagai *tag*/*bookmark* sehingga pada dasarnya semua metadata disimpan dalam satu repositori.
3. Desain repositori. Git menerapkan *Content Level* VCS dimana metadata yang direkam adalah delta perubahan dari setiap berkas. Sedangkan Mercurial menerapkan *File Level* VCS dimana setiap versi berkas disimpan secara utuh.

Git memiliki 139 fitur, sedangkan Mercurial memiliki 50 fitur. Namun, berdasarkan pada *Command Cheatsheet* untuk Git dan Mercurial, terdapat beberapa fitur yang biasa digunakan dalam kegiatan *versioning* yaitu :

Tabel 2.1 : Fitur Utama Git dan Mercurial

No	Git	Mercurial	Fungsi
1	*config*	*config*	Mengatur konfigurasi repositori seperti *username, email, remote*, dan lain-lain.
2	*init*	*init*	Membentuk repositori baru pada direktori.
3	*clone*	*clone*	Mengunduh repositori dari *remote repository*.
4	*status*	*status*	Melihat berkas yang telah diperbaharui sejak *commit* terakhir.
5	*diff*	*diff*	Melihat perbedaan yang terjadi setelah pembaharuan sejak *commit* terakhir.
6	*add*	*add*	Mendaftarkan berkas-berkas untuk di-*commit*
7	*reset*	*update*	Mengembalikan histori kepada suatu *commit* tertentu
8	*commit*	*commit*	Merekam sejarah perubahan pada *metadata versioning*
9	*branch*	-	Mengelola *branch*
10	*checkout*	-	Mengubah *branch* aktif
11	*merge*	*merge*	Sinkronisasi konten pada 2 *branch*
12	-	*annotate*	Melihat detail perubahan pada berkas
13	*rm*	*remove*	Menghapus berkas
14	*mv*	*move*	Memindahkan posisi berkas (digunakan juga untuk menamai ulang berkas)
15	-	*copy*	Menyalin berkas
16	*ls-files*	-	Menampilkan daftar berkas yang ada pada repositori
17	*stash*	*rollback*	Menyimpan informasi perubahan sementara dan mengembalikan status repositori ke *commit* terakhir
18	*log*	*log*	Menampilkan seluruh sejarah *commit*
19	*show*	*cat*	Menampilkan status perubahan berkas pada *commit* tertentu
20	*fetch*	-	Mengunduh semua sejarah dari *remote repository*
21	*pull*	*pull*	Sinkronisasi berkas dengan *remote repository*
22	*push*	*push*	Menyimpan sejarah perubahan ke *remote repository*

Untuk menguji pengaruh fitur terhadap *Configuration Item* (CI), dilakukan pengujian dengan membangun *shell script* yang berisi urutan perintah untuk mengeksekusi fitur-fitur *versioning* pada tabel di atas. *Shell script* dapat dilihat pada Lampiran 2. Dalam melakukan pengujian, terdapat beberapa fitur yang tidak diuji yaitu :

1. *config*, karena perintah ini digunakan untuk konfigurasi aplikasi git, bukan repositori.
2. *reset*, tidak diuji karena akan menghapus rekam sejarah *versioning*.
3. *clone*, karena memiliki fungsi sama dengan init.
4. *fetch*, karena memiliki fungsi yang sama dengan *pull* (*fetch* adalah fitur *pull* tanpa *merge*).

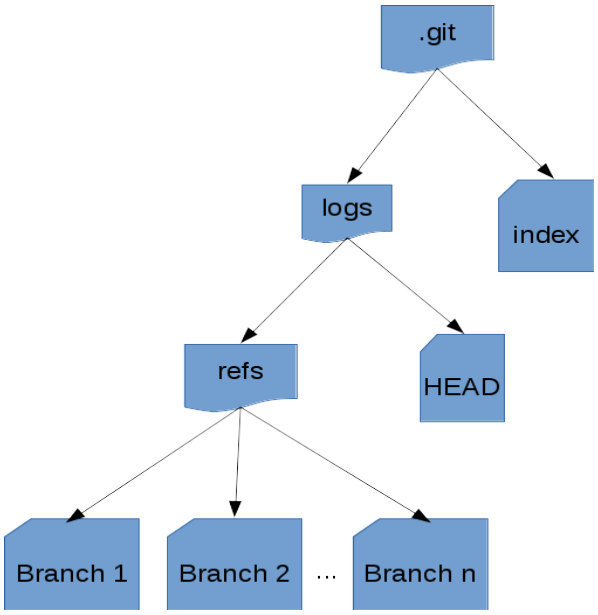
Dari hasil pengujian, terdapat beberapa fitur yang mengubah status CI yaitu :

No	Fitur	Pengaruh terhadap CI
1	*init/clone*	Membentuk CI
2	*add*	Mendaftarkan berkas yang akan direkam sebagai CI
3	*reset*	Mengubah histori CI ke *commit* sebelumnya
4	*commit*	Menambah rekam sejarah CI
5	*branch*	Mengelola *branch* untuk penyimpanan CI
6	*checkout*	Mengubah CI yang aktif ke *branch* tertentu
7	*merge*	Menyatukan CI pada *branch* berbeda
8	*rm*	Menghapus CI
9	*mv*	Memindahkan CI
10	*pull/push*	Sinkronisasi CI di repositori lokal dengan *remote repository*

Struktur Format *Log*

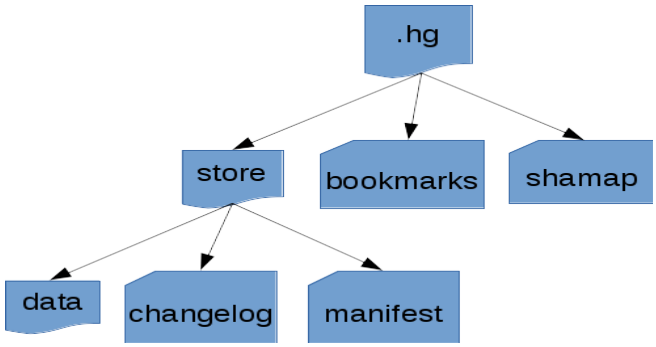
Semua aktifitas *versioning* akan direkam pada *log*. Setiap akifitas akan diberi id berupa 40 deret bilangan heksadesimal pada Git dan nomor urut serta dua belas deret bilangan heksadesimal pada Mercurial. Informasi mengenai aktifitas dibagi ke dalam dua bagian. Bagian pertama adalah rekam aktifitas *versioning* yaitu *commit*, *clone*, *pull*, *merge*, dan *push*. Pada bagian ini, direkam id aktifitas lama dan baru, siapa yang melakukan aktifitas, kapan aktifitas dilakukan, dan pesan aktifitas. Bagian kedua adalah rekam perubahan. Setiap id aktifitas memiliki rekaman perubahan yang disimpan dalam beberapa berkas. Aktifitas yang direkam adalah penambahan dan pengurangan kode dari setiap *commit* yang dilakukan. Namun informasi ini dienkripsi sehingga harus dilihat melalui aplikasi *versioning* itu sendiri. Rekam ini bisa dilihat dengan perintah “git show” pada Git dan “hg diff” pada Mercurial. Penambahan ditandai dengan karakter “+” dan pengurangan ditandai dengan karakter “-” pada setiap baris kode. Rekam perubahan dikelompokan berdasarkan id aktifitas yang merekam semua perubahan pada setiap berkas. Rekam perubahan ini adalah perubahan dari id lama ke id baru.

Berikut format *log* pada Git :



"*index*" adalah berkas rekaman semua perubahan yang dilakukan pengembang setiap kali melakukan *commit*. Id untuk setiap *commit* direkam di "HEAD". Pada berkas "HEAD", selain id, terdapat juga waktu *commit*, pelaku *commit*, dan pesan *commit*. Isi dari "HEAD" tergantung dari *branch* yang digunakan di komputer lokal. Rekam aktifitas *commit* pada tiap *branch* direkam pada direktori "refs" dengan penamaan berkas sama dengan nama *branch*.

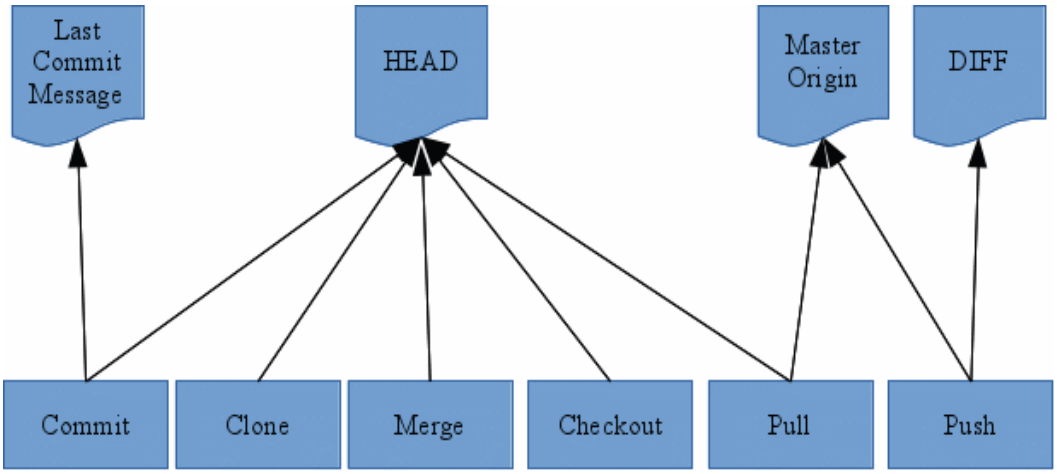
Berikut format *log* pada Mercurial :



"*bookmark*s" adalah rekam id untuk setiap *branch* yang di-*remote*. Id untuk setiap *commit* direkam pada "shamap". Pada "shamap" juga akan didefinisikan hubungan id Mercurial dengan Git jika *log* Mercurial adalah hasil konversi dari Git. Rekaman perubahan kode direkam pada direktori "store" dimana di dalamnya terdapat berkas "change*log*" dan "manifest" untuk merekam jejak perubahan, sedangkan direktori "data" menyimpan salinan kode sumber terakhir yang telah di-*commit*.

Struktur Berkas *Log Versioning*

Pada 2 kakas *versioning* yang dianalisis (Git dan Mercurial), semuanya memiliki ciri khas masing-masing dalam merekam jejak kegiatan *versioning*. Namun terdapat kemiripan dalam skema rekam jejak *versioning* seperti pada gambar berikut.



Tanda panah menunjukkan arah rekam jejak kegiatan *versioning*. Kegiatan versionung mulai dari *clone* hingga *push* direkam dalam beberapa berkas dengan sebuah hash sebagai identifikator. Artinya, setiap kegiatan yang dilakukan pengembang memiliki ID sendiri untuk nantinya digunakan sebagai identifikator dalam melakukan pelacakan balik seperti penelusuran sejarah perubahan suatu berkas atau sejarah *merge* antar *branch*. Pada dasarnya, ada 4 rekaman utama dalam ketiga kakas *versioning* yang dianalisis yaitu :

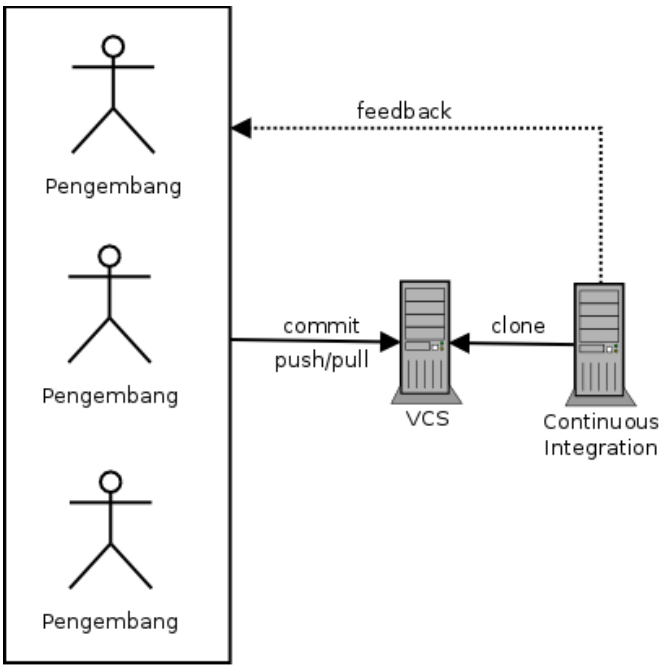
1. HEAD, semua kegiatan *remote* dalam *versioning* direkam disini mulai dari *clone* hingga *commit*.
2. Master Origin, semua kegiatan sinkronisasi kode sumber dari dan ke server direkam disini.
3. DIFF, semua perubahan yang terjadi direkam disini. Tetapi perubahan yang direkam adalah perubahan yang dilakukan pengembang itu sendiri.
4. Last *Commit Message*, merekam pesan dari *commit* terakhir yang digunakan untuk antisipasi *error* saat melakukan *push* secara *remote*.

Continuous Integration

Continuous Integration adalah praktek pengembangan yang mengharuskan pengembang untuk mengintegrasikan kode ke dalam repositori beberapa kali dalam satu hari. Setiap *check in* akan diverifikasi secara otomatis sehingga kesalahan dapat dideteksi sejak dini. Setiap integrasi diverifikasi oleh *build* otomatis (termasuk pengujiannya) untuk mendeteksi kesalahan integrasi dengan cepat (Fowler, 2006). Beberapa *best practices* dalam *Continuous Integration* adalah sebagai berikut :

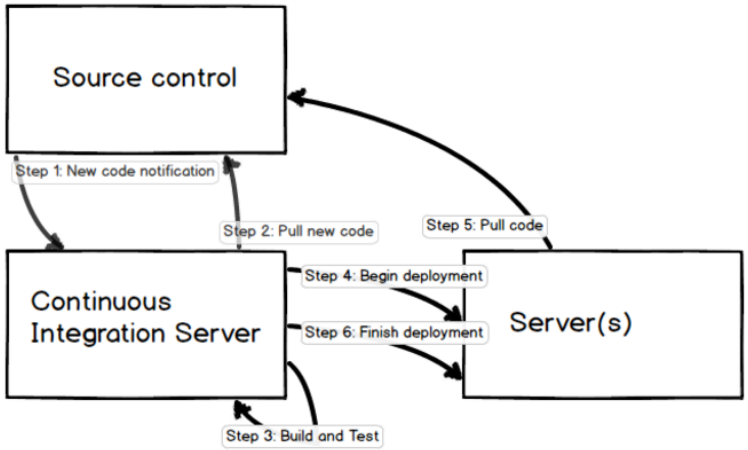
1. Setiap pengembang melakukan *build* terlebih dahulu di mesin masing-masing sebelum melakukan *commit* kode ke dalam VCS untuk memastikan bahwa perubahan yang dilakukan tidak menyebabkan kesalahan dalam proses *build*.
2. Pengembang melakukan *commit* setidaknya sekali dalam sehari.
3. *Build* terintegrasi dilakukan beberapa kali sehari di beberapa mesin yang berbeda.
4. Setiap *build* harus lulus uji sepenuhnya.
5. Produk yang dihasilkan bisa diuji fungsionalitasnya.
6. Memperbaiki *build* yang rusak adalah prioritas tertinggi.
7. Beberapa laporan pengembangan otomatis dilakukan saat melakukan *build*, seperti standar kode dan laporan analisis ketergantungan, untuk mencari hal-hal yang perlu ditingkatkan.

Dalam pengaplikasiannya, *Continuous Integration* tidak bisa lepas dari *versioning* karena aktifitas *build* dilakukan dengan mengambil kode sumber dari repositori dan menyalinnya ke dalam *workspace* sehingga siap untuk di-*build*. Aktifitas *build* pada *Continuous Integration* melibatkan semua fitur yang ada pada repositori sehingga harus dipastikan semua fitur dapat berjalan, baik berdiri sendiri maupun terintegrasi dengan fitur lain. Berikut keterhubungan antara *Continuous Integration* dengan VCS :



Dari gambar di atas, terlihat bahwa pengembang melakukan *commit* kode ke repositori. *Continuous Integration* melakukan *pull* ke repositori sehingga kode sumber tersalin ke *workspace Continuous Integration*. Setelah dilakukan *build* oleh *Continuous Integration*, maka feedback hasil *build* akan diberikan kepada pengembang seperti melalui email atau notifikasi lain. Standarnya, semua *build* dalam setiap jadwal *build* harus berhasil tanpa pesan *error*. Pengembang harus memastikan perubahan yang dilakukan terhadap fitur berkembang tanpa menimbulkan masalah pada kegiatan *build* yang dilakukan *Continuous Integration*.

Dalam prakteknya, CI tidak terlepas dari *versioning*. Gambar di bawah ini menunjukan bagaimana cara kerja CI yang berhubungan erat dengan *versioning*.



Dari gambar di atas terlihat bahwa server CI berhubungan dengan source control, dimana dalam prakteknya source control ini adalah repositori tempat pengembang melakukan *versioning*. Server CI akan melakukan *pull* ke repositori sehingga kode sumber tersalin ke dalam *workspace* lokal di server CI. Di dalam server CI akan dilakukan *automatic build* dan *automatic test* secara berulang sesuai jadwal yang sudah ditetapkan. Melalui proses berulang tersebut, akan dihasilkan kode sumber yang siap di-deploy. Server CI akan melakukan deploy secara otomatis dan mendistribusikan hasil deploy tersebut. Hasil deploy ini akan menjadi versi baru yang kembali disimpan dalam repositori *versioning*. Seluruh rangkaian pekerjaan ini akan terus berulang hingga perangkat lunak tidak diperbaharui lagi.

Dalam penelitian ini, repositori *versioning* adalah sumber data untuk menganalisis parameter kualitas dalam kegiatan *versioning*. Dalam kegiatan *versioning*, CI menjamin bahwa kode sumber dalam repositori sudah diuji dan bisa di-*build* tanpa menghasilkan *error*. Konsep *automatic build* pada CI akan diadaptasi pada penelitian ini untuk menguji kode sumber hasil kegiatan *versioning* pengembang.

Aktifitas Teknis *Versioning*

Kegiatan *versioning* dilakukan untuk merekam semua aktifitas yang dilakukan dalam siklus hidup pembangunan perangkat lunak. Dalam kegiatan *versioning*, pengembang perangkat lunak bergabung dalam satu tim mengembangkan perangkat lunak bersama-sama. Sejarah pengembangan masing-masing pengembang akan terekam sehingga jika terjadi kesalahan akan mudah untuk melakukan pelacakan balik. Namun pada prakteknya, kegiatan *versioning* yang dilakukan masing-masing pengembang tidak sama. Beberapa persoalan yang menyebabkan perbedaan pada kegiatan *versioning* adalah sebagai berikut :

1. Gaya *versioning* antar pengembang berbeda.
2. Pengalaman *versioning* masing-masing pengembang berbeda.
3. Minimnya standar yang ada mengenai kegiatan *versioning* yang baik.
4. Tidak jelasnya metrik pengukuran kualitas *versioning* yang dilakukan pengembang.

Berdasarkan pada aktifitas kegiatan *versioning* secara umum serta hasil eksplorasi *versioning tools*, dapat dilihat *command* umum dalam melakukan kegiatan *versioning* pada masing-masing *versioning tools*. *Command* umum ini berhubungan langsung dengan pemanfaatan *tools* VCS dan perintah apa saja yang dimanfaatkan terkait *workflow* kegiatan *versioning*. Berikut aktifitas *versioning* secara teknikal berdasarkan *workflow* kegiatan *versioning* :

No	Aktifitas	*Command* Umum	Mengubah status CI?
1	Membuat proyek baru	init	Ya
2	Meneruskan proyek lama	*clone*	Ya
3	Bekerja pada berkas	-	-
4	Periksa status	diff	Tidak
5	Mendaftarkan berkas	add	Ya
6	*Commit* perubahan	*commit*	Ya
7	Inspeksi sejarah *commit*	*log* show	Tidak
8	Membuat fitur baru	*branch*	Ya
9	Pindah ke fitur baru	checkout	Ya
10	Integrasi perubahan	*merge*	Ya
11	Pantau *remote branch*	checkout	Ya
12	Publish *branch* lokal	*push*	Ya
13	Pantau status perubahan	fetch	Tidak
14	Integrasi perubahan	*pull*	Ya
15	Upload perubahan	*push*	Ya

Berdasarkan tabel di atas tentang aktifitas teknikal *versioning* di atas, terdapat beberapa aktifitas yang berpengaruh terhadap CI. Dengan begitu, aktifitas *command* umum *versioning* dapat dijadikan referensi dalam menangkap parameter untuk menilai kualitas *versioning* yang dilakukan oleh pengembang.