سیستم مدیریت کد

در مدت ساخت یک برنامه گاهی اوقات نیاز است برنامه نویس به نسخه های قدیمی تر فایل های خود دسترسی داشته باشد تا بتواند آنچه را در قبل نوشته، مورد بازبینی قراردهد. برخی افراد این کار را با کپی کردن نسخه های قدیمی پروژه انجام می دهند که پس از بزرگ شدن پروژه و در کارهای تیمی مدیریت فایل ها به مشکل بر می خورد. لذا برای چنین مشکلاتی به سیستم مدیریت کد نیاز است.

وظیفه اصلی یک سیستم مدیریت کد:

وظیفه اصلی یک سیستم مدیریت کد ایجاد یک رویه خودکار در جهت دنبال کردن تغییرات فایل های پروژه است به طوری که این سیستم به ما بگوید هرفایل در چه زمانی، توسط چه کسی، به چه دلیل، چه تغییراتی(و ...) کرده است.

اهداف اصلی git

۱. سرعت بالا
۲. سادگی
۳. قدرت پشتیبانی بالا از merge/branching
۴. یک سیستم کاملاً توضیع شده
۵. قابلیت توسعه برای پروژه های بزرگ

تفاوت سیستم های مترکز و توزیع شده:

سیستم های کنترل نسخه از نظر معماری سیستم به دو دسته زیر تقسیم می شوند.

۱. VCS (Version Control System): سیستم های مدیریت نسخه متمرکز
۲. DVCS (Distributed Version Control System): سیستم های مدیریت نسخه توزیع شده

مفاهیم کلی مورد استفاده در git

Repository (مخزن) :

محلی که یک سیستم مدیریت نسخه از آن برای نگهداری تغییرات فایل استفاده می کند.

Repository در VCS ها: در VCSها مخزن ها به صورت متمرکز می باشند. به این معنا که مخزن برروی یک ماشین (سرور یا کامپیوتر شخصی) ذخیره شده و برنامه نویسان تغییرات فایل های خود را برروی آن می فرستند و این سرور وظیفه نگهداری تمام نسخه ها و اطلاعات مربوطه را از برنامه نویسان مختلف دارد و اشکال آن در این است که برنامه نویس در آن تنها به نسخه جاری که برروی سیستم خود قرار دارد دسترسی دارد و برای دسترسی به نسخه های قبلی باید به سرور دسترسی داشته باشد که ممکن است همیشه امکان دسترسی به سرور نباشد.

ام در DVCS علاوه بر یک مخزن برروی سرور که تمامی نسخه ها در آنجا هستند هر برنامه نویس یک نسخه محلی از مخزن در اختیار دارد. همچنین می توان با ایجاد یک sub Repository یک ساختار درختی ایجاد نمود که هر کدام از این زیر مخزن ها اطلاعات را در سرور اصلی قرار می دهند. به دلیل اینکه از مخزن چندید کپی وجود دارد backup گیری مطمئن تر است. از اشکالات این روش پیچیدگی پیاده سازی نسبت به VCS است.

Commit: بعد از اطمینان از صحت کد برای ثبت وضعیت فعلی باید فایل ها را commit کرد. با این کار یک نسخه جدید از فایل ها ایجاد می شود. به این ترتیب امکان بازگشت به این نقطه وجود دارد.

Pushing: بعد از commit کردن برنامه نویسان می خواهند فایل های خود را برای اعضای گروه اشتراک بگذارند. که این کار به وسیله عملیات pushing انجام دهند. Pushing عملی است که با استفاده از آن داده ها از یک Repository به Repository دیگر منتقل می شوند.

Upstream Repository: یک مخزن عمومی است که برای تمامی برنامه نویسان است که تغییرات فایل های خود را در آنجا push می کنند.

Pulling: pushing نصف عملیات بروز کردن کد ها است. در بسیار از موارد نیاز است تا آخرین تغییرات فایلها و آخرین بروز رسانی ها را نیز دریافت کنند. این کار در دو مرحله متفاوت انجام می شود.

۱- بازیابی داده ها از مخزن عمومی (fetch)
۲- الحاق داده های دریافت شده با داده های فعلی

معمولاً در بسیاری از سیستم های مدیریت کد این دو عملیات باهم نیاز است و با یک دستور هر دو کار انجام می شود که به مجموع این دو عملیات pulling می گویند.

Branch (شاخه): به ما این امکان را می دهد که بتوانیم برای قسمت های مختلف پروژه سوابق فایلی متفاوتی ایجاد کنیم. که البته معمولاً این شاخه ها زیاد با هم مرتبط نبوده و هرکدام مربوط به یک تیم کاری هستند. با این کار هرتیم کاری فقط تغییرات مربوط به فایل های خود را پیگیری کرده و در نهایت بعد از اطمینان از صحت کار خود آنرا در یک شاخه اصلی برای استفاده دیگر تیم ها قراردهد. در git شاخه پیشفرض master نام دارد. استاندارد کار این است که در شاخه master فایل های نهایی قرار گیرند.

Merging (ادغام): به عملیات ادغام دو یا چند شاخه با یکدیگر merging گویند. در این عملیات ممکن است مشکل conflict رخ دهد که git روش هایی برای مقابله با آنرا دارد.

Locking : با استفاده از این کار می توان مانع از تغییر یک فایل توسط برنامه نویسان دیگر شد که به دو صورت امکان پذیر است:

۱. Strict locking: در این روش پس از قفل گذاری فایل ها تنها همان کسی که فایل را قفل کرده امکان تغییر آنرا خواهد داشت. (در سیستم های توضیع شده توصیه نمی شود)
۲. Optimistic locking: در این روش هرکس تغییراتی در فایل انجام داد به گونه ای است که هنگام ادغام اختلالی رخ نمی دهد. یعنی وظیفه به عهده مصرف کننده است که آگاهی داشته باشد چگونه فایل را تغییر دهد. یعنی اینکه در حین تغییر فایل توسط ما شخص دیگری فایل را تغییر داده باشد و آنرا pull کرده باشد در زمان push فایل با خطا موجه می شویم. سیستم از ما می خواهد که ابتدا تغییرات فایل را pull کنیم و سپس فایل را push نماییم. در هنگام pull اگر برنامه نویسی قوانین تغییرات فایل را رعایت نکرده باشد ممکن است اعمال تغییرات با خطا مواجه شود.

Working tree (Directory): پوشه root است که فایل های پروژه در آن نگهداری می شوند. این پوشه باید حاوی پوشه ای به نام ".git" باشد که همان Repository ما است.

اشیاء در git : برای درک بهتر از عملکرد git باید اجزاء تشکیل دهنده آنرا بررسی کنیم. به طور کلی git دارای ۴ نوع object است که هرکدام وظیفه ای دارند.

۱. Tree: دقیقاً همانند دایرکتوری در سیستم فایل است. در واقع treeها ساختار درختی ایجاد می کنند تا وضعیت فایل ها و پوشه ها را در Repository حفظ نمایند. هر tree توسط یک کد منحصر به فرد SHA-1 نام گذاری می شود.
۲. BLOB (Binary Large Object): یک مجموعه از بایت ها که می تواند حاوی هر چیزی باشد (عکس، فایل متنی، فایل اجرایی و ...) در git فایل ها به صورت BLOB و به شکل کامل ذخیره

می شوند. همچنین مقدار هش شده محتویات فایل ها با استفاده از SHA-1 در خود فایل ذخیره می شود. پس اگر در فایل تغییری ایجاد شد مقدار هش شده فعلی با مقدار هش شده قبلی تفاوت دارد و git متوجه تغییر در فایل می شود.

نکته: در git برخلاف بسیاری از سیستم های مدیریت کد بعد از تغییرات فایل ها تنها تغییرات ذخیره نمی شود بلکه از تمام محتوا یک snapshot می گیرد. برای جلوگیری از افزایش حجم Repository، git تنها فایل هایی را که تغییر کرده اند را ذخیره کرده و به آنهایی که تغییر نکرده اند یک اشاره گر در snapshot جدید می افزاید.

۳. Commit: این object یک snapshot از وضعیت فعلی working tree است. در واقع با هر دستور commit این object ایجاد شده و حداقل حاوی اطلاعات زیر است:
   a. مقدار هش درختی که به ان اشاره می کند.
   b. نام ثبت کننده دستور commit
   c. توضیحی درباره علت ایجاد commit
   d. خود commit نیز توسط یک کد منحصر به فرد SHA-1 شناخته می شود.
۴. Tag: چون کارکردن با کدهای هش commit مشکل است می توان از تگ ها به عنوان نامی برای commit استفاده کرد. خود تگ می تواند حاوی توضیحاتی باشد.

(Stage (index: هر فایل قبل از ذخیره شدن در Repository توسط commit باید به stage آورده شود. در این حالت git تغییرات فایل را دنبال می کند و سپس می توان توسط دستور commit فایل را در Repository آورد.

ذخیره یک فایل در git دارای ۳ مرحله است.

a. Modified: فایل تغییر کرده اما به stage اضافه نشده است.
b. Staged: فایل تغییر کرده و به stage اضافه شده است.
c. Commited: فایل در Repository ذخیره شده است.

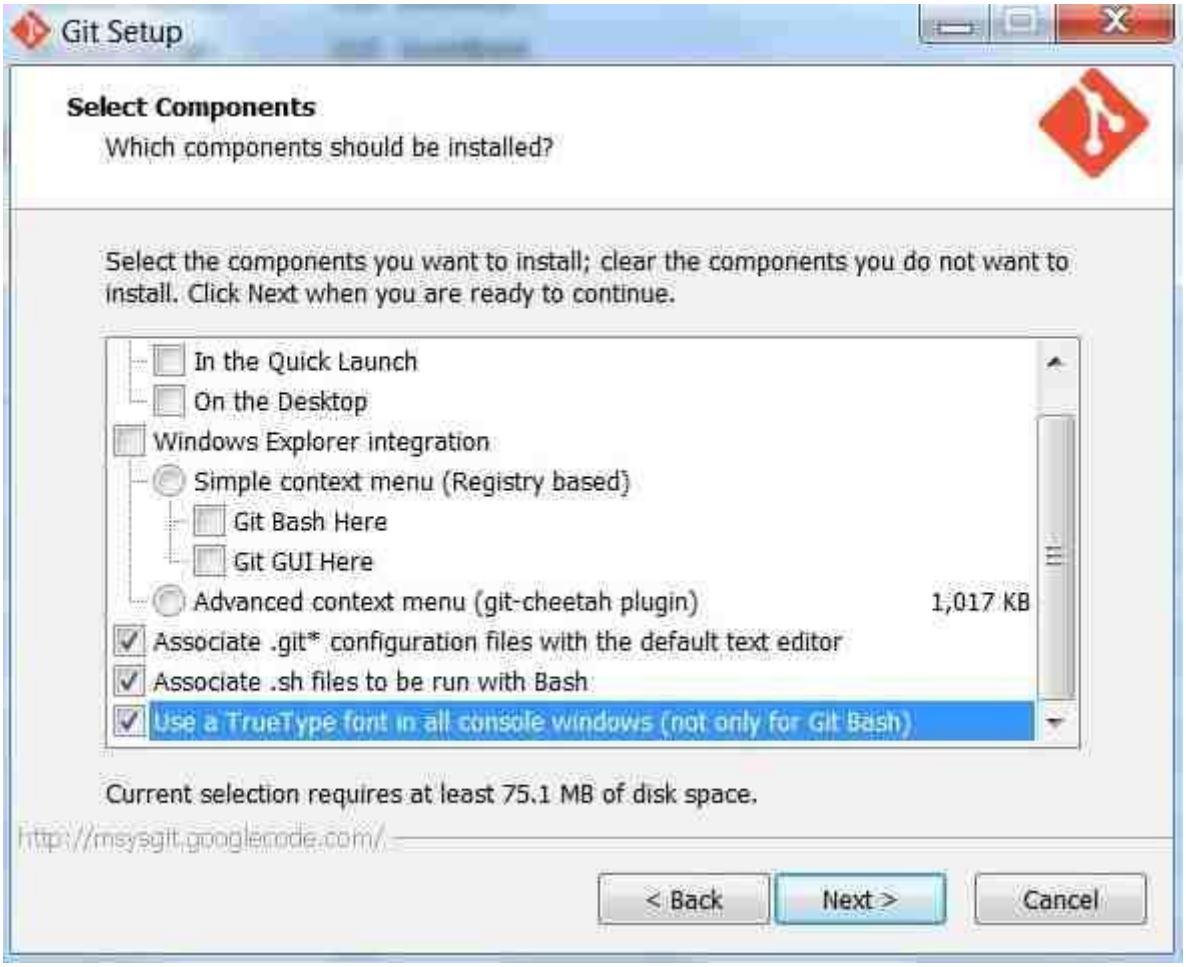
Head: اشاره گری است که با آخرین شئ commit اشاره می کند. هر Repository می تواند یک head برای شاخه های مختلف داشته باشد. اما در هر لحظه تنها یک head به عنوان head جاری شناخته می شود که معمولاً آنرا با حروف بزرگ 'HEAD' مشخص می کنند.

راه اندازی git :

برای نصب Git ابتدا به msysgit رفته و مطابق شکل زیر بر روی گزینه دانلود کلیک کنید. سپس در صفحه باز شده آخرین نسخه Git را دانلود نموده و فایل مربوطه را اجرا کنید:

شروع نصب:

در این مرحله بخش Windows Explorer Integration اهمیت دارد. در صورت انتخاب این بخش، بعد از نصب، Git Bashو Git GUI به منوی راست کلیک شما اضافه می‌شود. به این ترتیب با سرعت بیشتری می‌توانید به Git در یک پوشه خاص دسترسی داشته باشید.



در این مرحله از شما خواسته می‌شود تعیین کنید که آیا فقط می‌خواهید از طریق Git Bash با Git کار کنید یا با اضافه کردن فایل اجرایی Git به متغیرهای محلی ویندوز از طریق Command Prompt ویندوز نیز می‌خواهید به Git دسترسی داشته باشید. گزینه سوم هم Git و هم برخی از ابزارهای یونیکسی را به متغیرهای محلی اضافه می‌کند که سبب می‌شود شما یک خط فرمان قدرتمندتر در ویندوز داشته باشید. اما این کار ممکن است در برخی از برنامه‌های پیش فرض اختلال ایجاد کند بنابراین در انتخاب این گزینه احتیاط کنید.

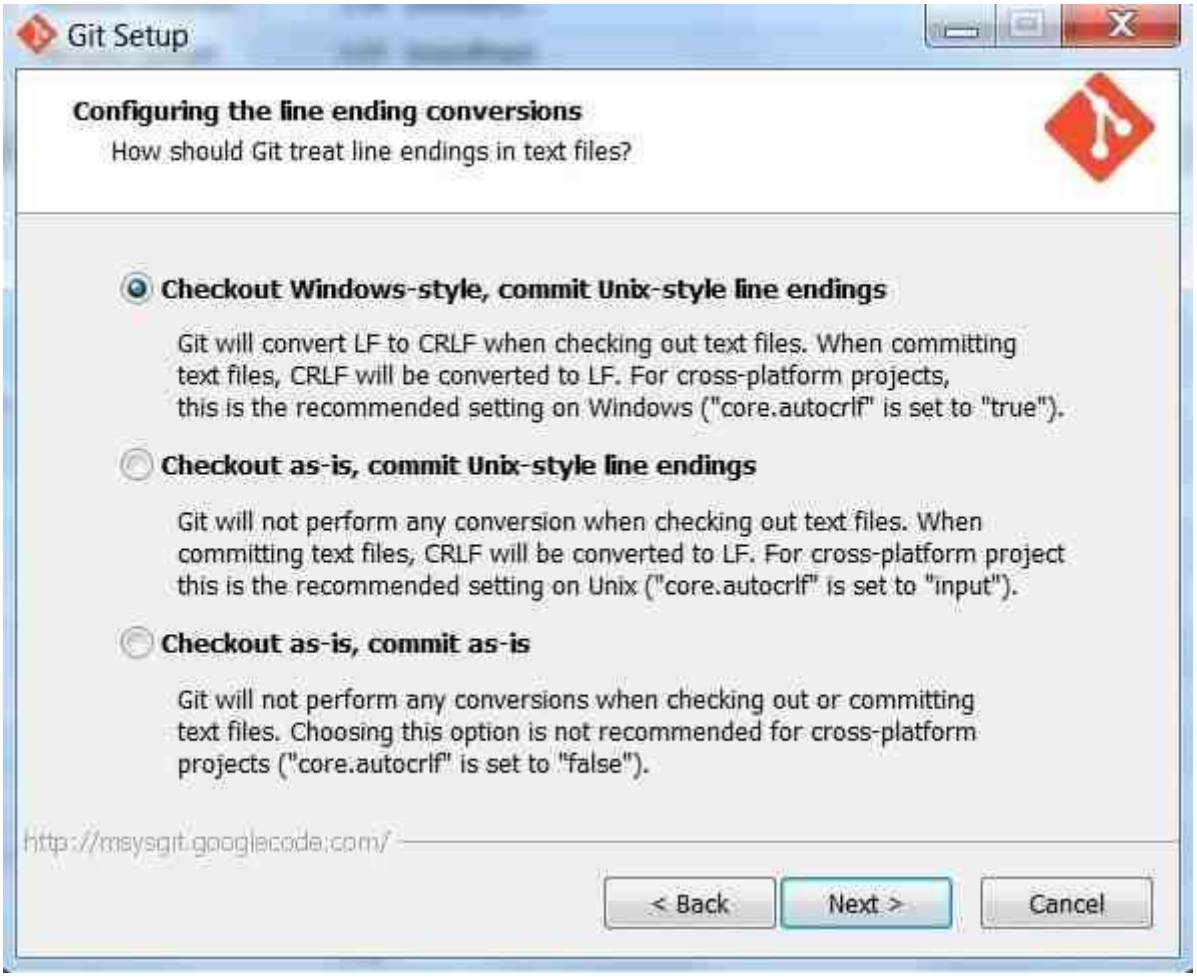
در این مرحله کاراکتری را که نشان دهنده انتهای خط است تعیین می‌کنید. این کاراکتر در ویندوز و یونیکس متفاوت است. بنابراین Git از شما می‌خواهد که برای حفظ سازگاری در محیط هایی که چند سیستمی هستند، آن‌را تعیین کنید.

گزینه اول به صورت فرمت یونیکس ذخیره و به شکل ویندوز بازیابی می‌شود (مناسب برای محیط ویندوز.(

گزینه دوم ذخیره به فرمت یونیکسی است و مناسب محیط‌های یونیکس است.

و آخرین گزینه فایل را بدون تغییر ذخیره و بازیابی می‌کند (از این گزینه نیز می‌توان هم برای Unix و هم windowsاستفاده کرد.(

بعد از این مرحله نصب آغاز می‌شود.



نکته :شما می‌توانید جهت دسترسی به یک محیط گرافیکی قوی از gitextensions استفاده کنید. با دانلود این فایل، هم خود Git و هم GUI هایی برای کارهای مختلف، نظیر مشاهده تفاوت‌های دو فایل یا نمایش گرافیکی شاخه‌ها به سیستم شما اضافه می‌شود.

پیکربندی git :

پس از نصب git باید آنرا پیکربندی کرد.

Git دارای سه نوع دسترسی برای پیکربندی است.

۱. سیستمی: این تنظیمات برروی کل سیستمی که git برروی آن نصب شده اعمال می گردد. این تنظیمات در محل نصب git در مسیر etc/gitconfig ذخیره می شود. برای تغییر این تنظیمات باید از دستور زیر استفاده کرد:

```
$ git config --system [parameters …]
```

در سطح کاربر: این تنظیمات در فایل config. در مسیر users/[username] ذخیره می شود و از دستور زیر برای تغییر آن استفاده می شود.

```
$ git config --global [parameters …]
```

در سطح Repository : برای هر پوشه Repository این فایل موجود است و اگر دستور git config را بدون سوییچ استفاده کنیم تغییرات برروی این فایل اعمال می شود.

```
$ git config [parameters …]
```

پس از نصب git باید دستورات و پیکربندی های زیر را انجام دهیم:

دستورات پیکربندی: هر commit دارای اطلاعات فردی است که این اطلاعات به این صورت تنظیم می شود.

نام کاربری:

```
$ git config --global user.name ‘user name’
```

ایمیل کاربر:

```
$ git config --global user.email ‘yourmail@example.com’
```

تنظیم رنگ پیام های git : پیشفرض auto

```
$ git config –global color.ui ‘color’
```

تعیین ویرایشگر متن پیشفرض برای git :

```
$ git config –global core.editor ‘your editor’
```

از این ویرایشگر می توان به طور مثال پس از دستور commit استفاده کرد تا دلیل commit مشخص شود.

نکته: نرم افزار ویرایشگر متن انتخابی باید در متغیرهای محلی ویندوز باشد. مثلاً notepad

```
$ git config –global core.editor notepad
```