Pro Git

حاوی هر آنچه لازم است شما درباره Git بدانید

پیشگفتار مترجم

دنیای امروز، دنیای نرم‌افزار است. و نرم‌افزار چیزی نیست که خود پدید آید. مجموعه‌ای از مهندسان با مهارت‌های مختلف، ابزارها و منابع متنوعی را در جهت تولید یک نرم‌افزار کارآمد و با کیفیت، به کار می‌گیرند. هر چه هماهنگی در تعاملات بین این مهندسان بیشتر باشد، نتیجه مطلوب‌تری حاصل می‌شود و نرم‌افزار، با کیفیت بالاتری تولید خواهد شد.

یکی از ابزارهای بسیار مهم که مهندسان نرم‌افزار در فرآیند تولید، از آن بهره‌مند می‌شوند، «سیستم کنترل نسخه» می‌باشد.

با قطعیت می‌توان گفت، یک تیم نرم‌افزاری بدون بهره جستن از این ابزار نمی‌تواند یک محصول بزرگ را به خوبی و با کیفیت بالا، تولید و نگهداری کند. اما استفاده نادرست و عدم تسلط توسعه‌دهندگان به این ابزار، گاهی اوقات می‌تواند ناهماهنگی‌های بزرگی ایجاد نماید و در پی آن، ساعت‌ها وقت، صرف ساماندهی وضعیت شود.

متاسفانه کم نیستند افراد متخصص در حوزه‌های مختلف تولید نرم‌افزار، که آشنایی عمیقی با ابزارهای کنترل نسخه ندارند و گاهی ابدا فلسفه وجودی آن را نمی‌دانند. این افراد به جهت تخصص خود در تیم‌های نرم‌افزاری استخدام می‌شوند، اما پس از گذشت زمان اندکی مشکلات زیادی را در تعامل با بقیه افراد تیم ایجاد می‌کنند.

به علاوه اکثر توسعه‌دهندگان نرم‌افزار به دلیل سهولت، رابط‌های کاربری گرافیکی را جهت استفاده از سیستم‌های کنترل نسخه برمی‌گزینند. تجربه نشان می‌دهد این افراد درک عمیقی از نحوه عملکرد سیستم کنترل نسخه ندارند. علت، این است که رابط‌های کاربری گرافیکی، تمامی قابلیت‌های این سیستم‌ها را پیاده‌سازی نمی‌کنند و علاوه بر این، کاربر، درگیر پشت صحنه نمی‌شود، لذا به لحاظ ایجاد درک از این سیستم‌ها، نتیجه مطلوبی در دراز مدت حاصل نخواهد شد.

کتاب حاضر که ترجمه کتاب Pro Git، ویرایش دوم، چاپ انتشارات Apress است، یکی از معروف‌ترین، روان‌ترین و جدیدترین کتاب‌های آشنایی با قدرتمندترین ابزار کنترل نسخه یا همان Git به حساب می‌رود. شیوه طبقه‌بندی کتاب طوری است که شما در پایان

هر فصل می‌توانید از ادامه مطالعه دست بکشید و به همان اندازه از ابزار قدرتمند Git بهره‌برداری نمایید. همچنین در این کتاب تمرکز اصلی بر روی استفاده از command ها می‌باشد و رابط‌های کاربری گرافیکی، تنها به صورت بسیار خلاصه معرفی می‌شوند.

در ترجمه حاضر، برای هر واژه تخصصی در اولین محل وقوع، معادل زبان اصلی به صورت پاورقی ذکر شده است. علاوه بر این، چون ممکن است خواننده در محل‌های دیگری نیز که واژه‌ی مزبور به کار رفته نیازمند مراجعه به واژه‌ی اصلی باشد، واژه‌نامه‌ای در انتهای کتاب آمده است.

در گزینش معادل فارسی واژه‌های تخصصی دو نکته را مورد توجه قرار داده‌ایم:

۱. یکی از ملاک‌های اصلی معادل‌سازی، میزان رواج آن در میان کتب و اهل فن بوده است.
۲. واژه‌هایی که معادل متداولی در فارسی نداشته‌اند و معادل‌سازی برای آنها، شیوایی ترجمه را دچار آشفتگی می‌سازد، عینا در متن آورده شده‌اند.

عنایت داشته باشید، پاراگراف‌هایی که در کادر قرار گرفته‌اند، اضافات مترجم به متن اصلی می‌باشد.

امیدوارم خواندن این کتاب برای شما بسیار سودمند باشد.

تقدیم‌نامه

تقدیم به همسر مهربانم که حقیقتا پشتیبان من در تمامی اهدافم بوده است.

پیشگفتار نویسنده اول[1]

به ویرایش دوم Pro Git خوش آمدید. اولین ویرایش در اواخر سال ۲۰۰۹ منتشر شد. پس از انتشار اولین ویرایش، تغییرات زیادی در Git ایجاد شده است. البته باید گفت بسیاری از اصول Git هنوز دچار تغییر نشده‌اند و این موضوع، خود یک نقطه قوت به حساب می‌آید. در حالی که اکثر دستورها و مفاهیم، امروز همچنان معتبر هستند (و این به لطف تیم اصلی Git است که در حفظ سازگاری با نسخه‌های قدیمی[2] بسیار فوق‌العاده عمل می‌کنند) اضافات و تغییرات قابل توجهی پیرامون Git به وجود آمده است. نسخه‌ی دوم این کتاب قصد دارد تا این تغییرات را پوشش دهد و کتاب را به روزرسانی کند. بنابراین برای خوانندگان جدید، بسیار مفیدتر از اولین ویرایش است.

وقتی من اولین ویرایش را نوشتم، استفاده از Git هنوز رایج نشده بود و به سختی توسط توسعه‌دهندگان پذیرفته می‌شد. بعضی از تیم‌های خاص شروع به استفاده از آن کرده بودند اما به هیچ وجه در موقعیت کنونی قرار نداشت. رفته رفته محبوبیتش در میان جامعه توسعه‌دهندگان متن‌باز[3]، زیاد شد. Git قدم به سیستم عامل Windows گذاشت و به سرعت در حوزه رابط‌های گرافیکی و پشتیبانی از IDE های مختلف، پیشرفت نمود. ویرایش اول Pro Git هیچ کدام از این موضوعات را پوشش نداده است. لذا یکی از اهداف اصلی این ویرایش، بحث و بررسی مرزهای جدید در Git خواهد بود.

جامعه توسعه‌دهندگان متن‌باز نیز در طول این سال‌ها به لطف Git رشد کرده است. در ابتدای کار، حدود هفت سال پیش، وقتی شروع به نوشتن این کتاب کردم، مدت کمی بود که در یک شرکت بسیار مشهور، شروع به کار کرده بودم. این شرکت مشهور، یک وب‌سایت با میزبانی Git با نام Github را توسعه می‌داد. در زمان انتشار شاید تنها چند هزار نفر از مردم در حال استفاده از سایت بودند و تنها چهار نفر از ما روی آن کار می‌کردیم. همان طور که مشغول نوشتن این مقدمه هستم، تعداد پروژه‌های میزبانی شده

---

[1] Scott Chacon، نویسنده کتاب، یکی از فعالان حوزه ترویج تکنولوژی Git می‌باشد. وی به عنوان توسعه‌دهنده Ruby در سال ۲۰۰۸ با شرکت Github.com شروع به همکاری نموده است.

[2] Backward compatibility

[3] Open-Source Communities

در Github به ۱۰ میلیون رسیده است. ۵ میلیون کاربر و ۲۳۰ کارمند دارد. چه بخواهیم چه نخواهیم، Github به شدت مرزهای فعالیت جوامع متن‌باز را نسبت به زمانی که اولین ویرایش را می‌نوشتم، تغییر داده است.

آمار اعلام شده توسط Github در سال ۲۰۱۷ برای تعداد پروژه‌های میزبانی شده، بیش از ۵۳ میلیون است. و بیش از ۲۰ میلیون کاربر و ۵۹۷ کارمند دارد.

در ویرایش اول این کتاب بخش بسیار کوچکی را به Github اختصاص داده بودم که این موضوع برایم خوشایند نبود. اهمیت Github در جامعه Git، اجتناب‌ناپذیر است. لذا تصمیم گرفتم تا آن بخش از کتاب را به بیان عمیق این مسئله که Github چیست و چگونه به طور مؤثر از آن استفاده کنیم تبدیل کنم.

اگر قصد آموختن Git را دارید باید بدانید چگونه استفاده کردن از Github، فارغ از اینکه بخواهید آن را برای میزبانی پروژه‌های خود انتخاب کنید یا نه، بسیار برای شما ارزشمند است. این مسئله به شما کمک می‌کند تا در جامعه توسعه‌دهندگان، گام بردارید و این فرصت را به شما می‌دهد تا بقیه را بشناسید و بقیه نیز توانایی‌های شما را بشناسند. تغییر بزرگ دیگر از زمان آخرین انتشار این کتاب، توسعه و اضافه شدن پروتکل HTTP برای تراکنش‌های شبکه‌ای Git بوده است. اکثر مثال‌ها در کتاب جهت سادگی از SSH به HTTP تغییر داده شده‌اند.

امیدوارم از این کتاب نهایت استفاده را ببرید.

پیشگفتار نویسنده دوم[۴]

باید اعتراف کنم اولین ویرایش این کتاب همان چیزی بود که زندگی حرفه‌ای مرا به Git گره زد. خواندن این کتاب مقدمه‌ای شد برای من تا سبک ملموس و شیرین‌تری را در تولید نرم‌افزار تجربه کنم. قبل از آشنایی با این کتاب، چندین سال مشغول به کار در این حرفه بودم، اما این کتاب مسیر به شدت جذابی را به من نشان داد که قبلا نمی‌شناختم.

اکنون، بعد از چند سال، من یکی از توسعه‌دهندگان Git به شمار می‌روم. من برای بزرگ‌ترین شرکت میزبانی Git کار کرده‌ام، و به سراسر جهان جهت برگزاری کارگاه‌های آموزشی Git سفر کرده‌ام. وقتی Scott پرسید آیا علاقه‌مند به کار کردن بر روی ویرایش دوم هستم، در پذیرفتن آن، لحظه‌ای درنگ نکردم.

کار کردن بر روی این کتاب برای من، لذت و امتیاز بزرگی بوده است. امیدوارم به همان اندازه که برای من مفید بوده است برای شما نیز مفید واقع شود.

[۴] Ben Straub، از سال ۲۰۱۲ تا ۲۰۱۴ در شرکت Github بر روی Libgit2 و نسخه تحت ویندوز Github مشغول به فعالیت بوده است.

مقدمه

در اینجا یک خلاصه‌ی کوتاه از ده فصل و سه ضمیمه این کتاب را ارائه می‌دهیم. در مورد هر کدام از اصطلاحات مطرح شده در این مقدمه، به تفصیل در بخش مربوطه خواهید خواند.

در **فصل ۱**، سیستم‌های کنترل نسخه (VCSs) و مبانی Git را پوشش خواهیم داد. در این فصل به مسائل فنی نمی‌پردازیم. درباره اینکه Git چیست، چرا در سرزمینی پر از VCSها حضور پیدا کرده، چه چیز آن را مجزا می‌کند، و چرا افراد زیادی در حال استفاده از آن هستند، خواهیم خواند. و در نهایت برای کسانی که آن را بر روی کامپیوتر خود نصب ندارند توضیح خواهیم داد که چگونه آن را دانلود و برای اولین بار نصب کنیم.

در **فصل ۲**، استفاده ابتدایی از Git را بیان خواهیم کرد. مطالب این فصل، ۸۰٪ از مواردی که شما به صورت روزمره در Git با آن سروکار دارید را پوشش می‌دهد. بعد از خواندن این فصل، شما باید قادر باشید تا یک repository را clone کنید. تاریخچه‌ی پروژه را مرور نمایید. فایلها را تغییر دهید، و تغییرات خود را commit کنید.

فصل ۳، در مورد مدل ایجاد شاخه[۵] در Git است، که یکی از ویژگی‌های فوق‌العاده Git به حساب می‌آید. در این فصل شما متوجه می‌شوید که چه چیزی Git را از بقیه‌ی VCS ها متمایز کرده است. اگر اولین بار باشد که با این مفهوم آشنا می‌شوید، شاید پیش خود فکر کنید که چطور تا به حال بدون این قابلیت، پروژه‌های خود را انجام داده‌اید.

فصل ۴، در مورد نصب و راه‌اندازی Git بر روی یک سرور، توضیح خواهد داد. این فصل برای کسانی است که می‌خواهند Git را در داخل سازمان خود بر روی یک سرور، راه‌اندازی نمایند. البته در این فصل گزینه‌هایی را معرفی می‌کنیم که این سرویس را به شما ارائه می‌دهند و دیگر شما نیازی به راه‌اندازی سرور ندارید. در واقع این سرویس‌ها Git را برای شما میزبانی[۶] می‌کنند.

در **فصل ۵**، در مورد انواع workflow های توزیع شده و نحوه پیاده‌سازی آنها با Git،

---

[۵] branching

[۶] host

توضیح خواهیم داد. در پایان این فصل، شما قادر خواهید بود پروژه خود را به چندین remote repository متصل نمایید و همچنین میاموزید که چگونه به وسیله پست الکترونیکی با Git کار کنید. این فصل اولین قدم برای این است که شما توانایی‌هایتان را به دیگران نشان دهید. شما خواهید توانست بر روی پروژه‌های متن‌باز تغییراتی را جهت بهبود آنها، با نام خود ایجاد نمایید. این موضوع در غنی‌سازی رزومه حرفه‌ای شما بسیار مؤثر است.

فصل ۶ به طور مفصل، سرویس GitHub را پوشش می‌دهد. از ثبت‌نام و مدیریت حساب، ایجاد و استفاده از repository های Git، روال مشارکت کردن[۷] در دیگر پروژه‌ها و یا پذیرفتن مشارکت دیگران در پروژه‌های شما، تا مفاهیم مربوط به hook، API و به طور کلی بسیاری از نکات ریزی که زندگی حرفه‌ای شما را آسان می‌کند.

فصل ۷ درباره دستورات پیشرفته Git است. در پایان این فصل، شما به دستور reset مسلط خواهید شد. دستوری که اکثر کاربران از آن وحشت دارند. در مورد استفاده از جستجوی دودویی برای شناسایی باگ‌ها، ویرایش تاریخچه[۸]، گزینش ویرایش‌ها، و بسیاری مسائل دیگر خواهید آموخت.

فصل ۸ درباره پیکربندی[۹] محیط Git شما است. در این فصل با hook و کاربردهای آن آشنا می‌شوید و استفاده از آن را خواهید آموخت.

در **فصل ۹** با Git و دیگر VCSها سر و کار خواهید داشت. در این فصل می‌آموزید چگونه از VCSهای دیگر به Git مهاجرت کنید. سازمان‌های زیادی همچنان از SVN استفاده می‌کنند و در مقابل تغییر به شدت مقاومت می‌کنند، اما در این فصل شما قدرت باور نکردنی Git را جهت مهاجرت از Subversion به Git خواهید دید.

فصل ۱۰، وقتی به این فصل می‌رسید، همه چیز را در مورد Git می‌دانید و می‌توانید با قدرت و ظرافت از آن، بهره‌برداری نمایید. اما می‌توانید باز هم جلوتر روید. این فصل وارد جزییات پیچیده، اما در عین حال بسیار زیبای Git می‌شود. اگر علاقه‌مندید بدانید

---

[۷] Contribute

[۸] Editing history

[۹] Configuring

Git چگونه اشیاء خود را ذخیره می‌کند، مدل اشیاء[۱۰] آن چگونه است، packfile چیست و یا می‌خواهید در مورد پروتکل‌های سرور Git، اطلاعات بیشتری کسب کنید، خواندن این فصل توصیه می‌گردد. این موضوع کاملا به کنجکاوی خود شما در مورد Git بستگی دارد.

در **ضمیمه الف** در مورد تعدادی از GUI های معروف Git و نحوه استفاده از آن در چند IDE مشهور و پرکاربرد توضیحاتی ارائه خواهیم نمود. و به شما خواهیم گفت، چه چیزهایی در این محیط‌ها برای شما قابل دسترس است و از طرفی چه محدودیت‌هایی خواهید داشت. در این ضمیمه یک مرور کلی و سریع به نحوه استفاده از Git در محیط‌هایی مانند shell ، Visual Studio و Eclipse خواهیم داشت.

در **ضمیمه ب** script نویسی و توسعه Git از طریق ابزارهایی مثل libgit2 و JGit را مورد بررسی قرار می‌دهیم. اگر شما علاقه‌مند به نوشتن ابزارهای پیچیده و سریع برای بهره‌برداری از Git هستید و به دسترسی سطح پایین Git نیاز دارید، این ضمیمه، جایی است که در آن چشم‌انداز خوبی در مورد این مسائل پیدا خواهید کرد.

در نهایت در **ضمیمه پ** ما تمام دستورات عمده Git را، یکی یکی مرور خواهیم کرد و اشاره می‌کنیم که در کجای کتاب آن مبحث را پوشش داده‌ایم. اگر به دنبال توضیح در مورد command خاصی هستید می‌توانید با مراجعه به این ضمیمه، آدرس توضیحات تکمیلی در کتاب را بیابید.

---

[۱۰] Object Model

فهرست مطالب

فصل ۱ – گام نخست

در این فصل کار با Git را شروع می‌کنیم. با توضیح در مورد برخی زمینه‌ها در مورد «ابزارهای کنترل نسخه» آغاز خواهیم کرد. سپس با این سوال که چگونه گیت را بر روی سیستم خود راه‌اندازی کنیم ادامه خواهیم داد. در پایان این فصل شما باید فلسفه وجودی گیت را بیابید، بدانید چرا استفاده از گیت لازم است و چرا شما باید مهارت استفاده از آن را پیدا کنید.

درباره کنترل نسخه [۱۱]

«کنترل نسخه» و علت اهمیت آن چیست؟ کنترل نسخه سیستمی است که تغییرات یک فایل یا مجموعه‌ای از فایل‌ها را در بازه زمان ثبت میکند به طوری که شما میتوانید نسخه‌های خاصی را بعدا مجددا بازیابی نمایید. در مثال‌های این کتاب نسخه فایل‌های مربوط به کدهای نرم‌افزاری تحت کنترل در می‌آیند. اما باید بدانید در واقع نسخه انواع فایل بر روی کامپیوتر با «ابزار کنترل نسخه» قابل کنترل است.

اگر شما یک طراح گرافیک هستید و می‌خواهید هر نسخه از یک تصویر یا طرح را نگه دارید (که قطعا مایل به انجام این کار هستید)، استفاده از یک سیستم کنترل نسخه یا VCS [۱۲] یک اقدام زیرکانه است. این سیستم به شما این امکان را می‌دهد تا فایل‌های مشخص یا کل پروژه را مجددا به حالت قبلی بازگردانید، تغییرات را در طول زمان مقایسه کنید، ببینید چه کسی آخرین تغییرات را انجام داده است که ممکن است مشکل‌ساز شده باشد و غیره. به طور کلی در صورت استفاده از VCS ها دیگر مفهومی تحت عنوان از بین رفتن تغییرات و یا گم شدن فایل‌ها وجود ندارد. و در صورت بروز هرگونه خلل و مشکل در فایل‌ها شما به سادگی با استفاده از این ابزار با کمترین سربار می‌توانید فایل‌ها را بازیابی نمایید.

---

[۱۱] Version Control

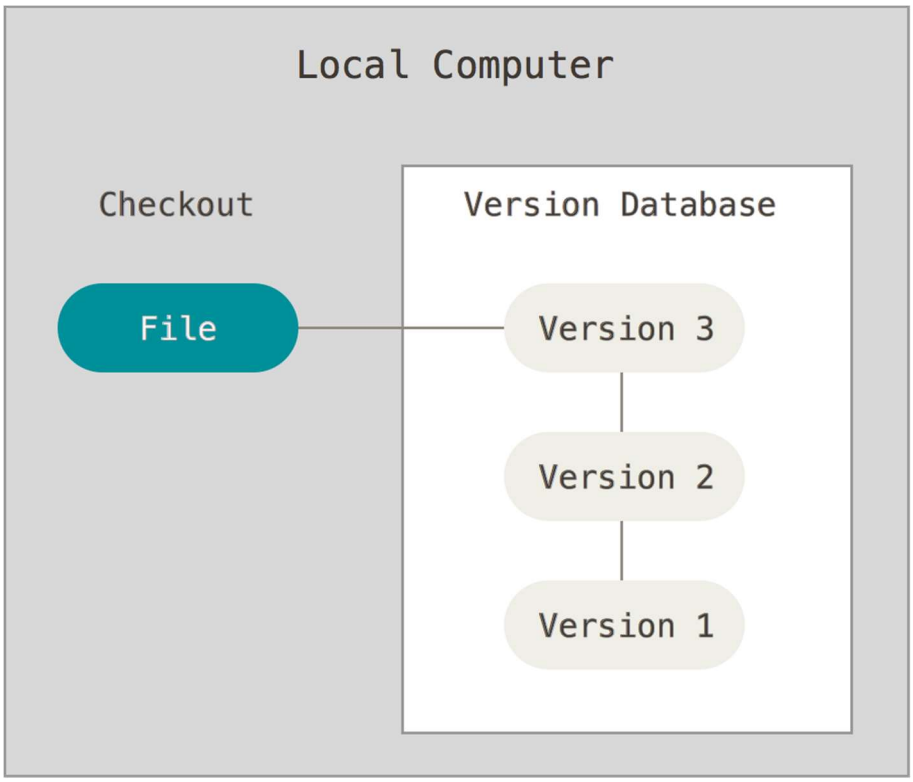
[۱۲] Version Control System

سیستم‌های کنترل نسخه محلی

روش انتخابی بسیار از مردم برای کنترل نمودن نسخه کپی کردن فایل‌های یک پروژه در یک پوشه دیگر است. زکاوتی که در این روش می‌توان اتخاذ نمود این است که نام پوشه جدید را طوری انتخاب نمایند که زمان و تاریخ ایجاد در آن نهفته باشد. این روش به دلیل ساده بودن بسیار رایج است. اما در عین حال به شدت در معرض خطا و اشتباه می‌باشد. فراموش نمودن این که شما در کدام پوشه یا در واقع کدام نسخه از کارتان قرار دارید و این که تصادفا در یک فایل از نسخه دیگری که مد نظرتان نیست اشتباها تغییراتی ایجاد نمایید بسیار محتمل است.

برای مقابله با این مسائل، برنامه‌نویسان مدت‌ها قبل سیستمی را طراحی نمودند تحت عنوان «سیستم‌های کنترل نسخه محلی» [۱۳] که در این سیستم‌ها یک پایگاه داده‌ی ساده، وظیفه نگهداری تمام تغییرات فایل‌ها را به عهده داشت.

[۱۳] Local Version Control Systems



شکل ۱- کنترل نسخه محلی

یکی از ابزارهای مشهورتر VCS سیستمی با نام RCS [۱۴] بود، که همچنان با بسیاری از سیستم عامل‌های امروزی ارائه می‌شود. حتی وقتی شما Developer Tools را بر روی سیستم عامل Mac OS X نصب و فعال نمایید به همراه خود RCS نیز نصب می‌شود و دستورات آن قابل اجرا خواهد بود. RCS با نگهداری مجموعه patch ها (یا به عبارت دیگر مجموعه‌ای از اختلاف فایل‌ها) در یک فرمت خاص بر روی دیسک کار میکند. و با این ساز و کار می‌تواند در هر نقطه از زمان با در کنار هم گذاشتن همه patch ها دوباره فایل را به همان صورت که بود ایجاد کند.

[۱۴] Revision Control System

سیستم‌های کنترل نسخه متمرکز

مسئله عمده دیگری که توسعه‌دهندگان [۱۵] با آن مواجه می‌شوند این است که آن‌ها نیاز دارند بر روی یک پروژه با یکدیگر همکاری داشته باشند. در VCS های محلی همان طور که گذشت پایگاه داده بر روی همان کامپیوتر توسعه‌دهنده قرار دارد لذا توسعه‌دهندگان دیگر نمی‌توانند از تغییرات فایل‌ها اطلاع داشته باشند. و در واقع همین مسئله باعث می‌شوند تا VCS های محلی ناکارآمد شوند.

لطفا کمی درنگ کنید. شما برای حل این مسئله چه راه حلی پیشنهاد می‌کنید؟

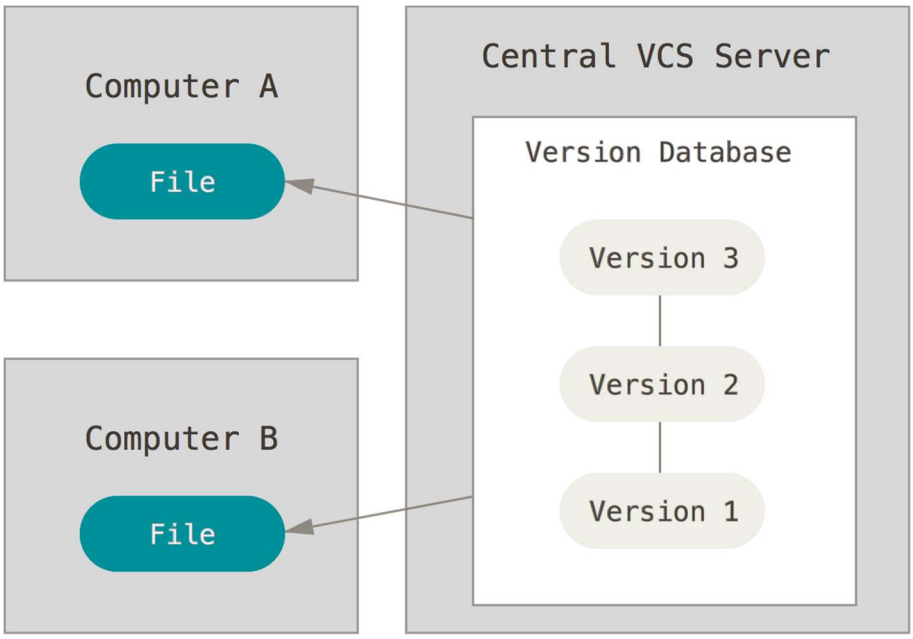
برای حل این مسئله، سیستم‌های کنترل نسخه متمرکز (CVSs) [۱۶] طراحی شدند. این سیستم‌ها، از قبیل CVS، Subversion و Perforce یک سرور مرکزی دارند که شامل همه فایل‌ها و نسخه‌هایشان و همچنین client هایی که فایل‌ها را از روی سرور مرکزی می‌خوانند و یا تغییر می‌دهند می‌شود. چندین سال، این ساز و کار یک استاندارد برای کنترل نسخه بود.

این ساز و کار مزایای زیادی ارائه میدهد، مخصوصا در مقایسه با VCSهای محلی. برای مثال، تمام توسعه‌دهندگان به دقت می‌دانند که افراد دیگر در حال انجام چه کاری بر روی پروژه هستند. مدیران پروژه می‌توانند تمام دسترسی‌ها برای دیگران و هر آنچه که دیگران باید انجام دهند را کنترل نمایند. و البته باید گفت مدیریت کردن و نگهداری یک CVCS با یک پایگاه داده مرکزی بسیار آسان‌تر و عملی‌تر از سر و کار داشتن با تعداد زیادی پایگاه داده بر روی هر یک از client ها خواهد بود.

---

[۱۵] Developers

[۱۶] Centralized Version Control Systems



شکل ۲- کنترل نسخه مرکزی

اگر چه این ساز و کار یک سری معایب جدی نیز دارد. حتما به ذهن شما نیز خطور کرده است. اگر سرور مرکزی به هر علت از کار بیفتد چه پیش خواهد آمد؟ درست است پاشنه آشیل این سیستم و آشکارترین عیب این ساز و کار صحت سرور مرکزی می‌باشد. اگر آن سرور برای یک ساعت خراب شود، آنگاه در طول آن یک ساعت هیچ کس به هیچ عنوان نمیتواند تغییرات خود را بر روی سرور منتقل نماید، یا حتی نمی‌توان تغییرات جدیدی بر روی فایل‌ها ایجاد نمود و یا تغییرات دیگران را از سرور مرکزی دریافت نمود. اگر هارد دیسکی که پایگاه داده‌ی مرکزی بر روی آن است خراب شود، و شما backup مناسب از اطلاعات نداشته باشید، می‌توان گفت شما تقریبا همه چیز را از دست داده‌اید. تاریخچه‌ی کامل پروژه به جز آخرین نسخه از پروژه که بر روی کامپیوتر هر یک از client ها قرار دارد از بین خواهد رفت. سیستم‌های VCS محلی نیز دچار همین مشکل هستند- وقتی شما کل تاریخچه‌ی پروژه را تنها در یک محل ذخیره نمایید، خطر از دست دادن اطلاعات به شدت وجود دارد.

شما نیز حتما در ذهن خود به راه حل می‌اندیشید! شاید راه حلی که در ذهن شما وجود دارد بسیار ساده باشد. و البته واقعیت نیز همین است. تئوری راه حل بسیار ساده است.

سیستمهای کنترل نسخه توزیع شده

اینجا جایی است که سیستم‌های کنترل نسخه توزیع شده (DVCSs) [۱۷] قدم در میدان می‌گذارند تا تمامی مشکلات ساز و کارهای پیشین را حل نمایند. در DVCS (مانند Git، Mercurial، Bazaar، Darcs) در واقع تمامی client ها نقش سرور مرکزی در CVCS ها را بازی می‌کنند. بدین صورت که هر client فقط نسخه نهایی را از روی سرور اصلی نمی‌گرد. بلکه در حقیقت هر client یک clone کامل از repository [۱۸] است که تمامی تاریخچه‌ی فایل‌ها را شامل می‌شود. از این رو اگر سروری دچار مشکل شود، repository هر یک از client ها می‌تواند به عنوان backup استفاده شود تا اطلاعات سرور اصلی بازیابی گردد. هر clone واقعا یک backup کامل از تمامی اطلاعات است.

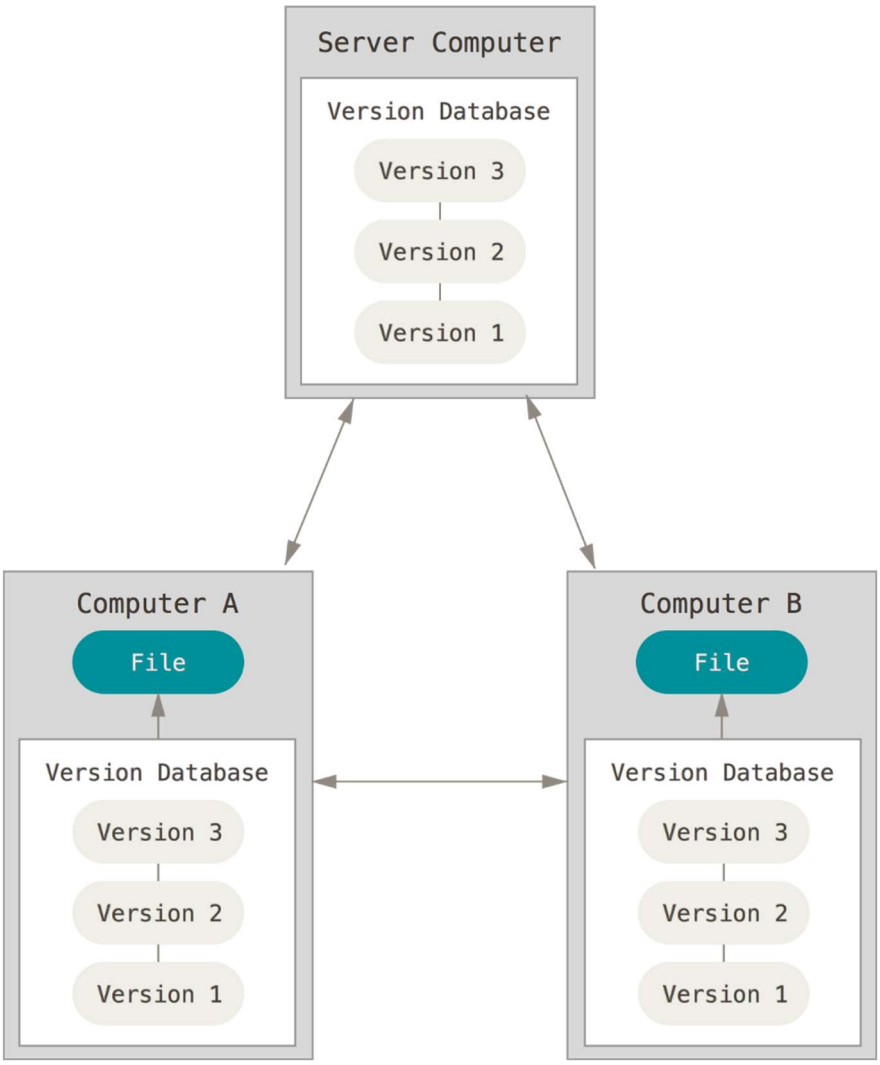
بـــه علاوه بسیـاری از ایـن سیستـــم‌ها (DVCS) می‌تـوانند به خوبـی با چندین remote repository [۱۹] کار کنند. این ویژگی این امکان را برای شما فراهم می‌سازد تا به صورت همزمان با چندین گروه از توسعه‌دهندگان بر روی یک پروژه واحد اما با رویکردها و workflow های متفاوت کار کنید که این امکان در سیستم‌های متمرکز به هیچ عنوان وجود ندارد.

---

[۱۷] Distributed Version Control Systems

[۱۸] اصطلاحا به ساختار داده‌ای می‌گوییم که تمامی فایل‌ها و اطلاعات مربوط به تغییرات آن‌ها را در طول زمان نگهداری می‌کند.

[۱۹] در فصل پنجم با مفهوم remote repository آشنا خواهید شد.



شکل ۳-کنترل نسخه توزیع شده

تاریخچه‌ی کوتاهی از Git

همانند بسیاری از مسائل بزرگ در زندگی، انگیزه و محرک ایجاد Git پدیده تخریب خلاق[۲۰] بود.

هسته سیستم‌عامل لینوکس یک پروژه نرم‌افزاری متن باز بزرگ است. اکثر طول عمر حفظ و نگهداری هسته لینوکس (۱۹۹۱-۲۰۰۲) تغییرات در نرم‌افزار به صورت patch ها و فایل‌های آرشیو شده منتشر شدند. در سال ۲۰۰۲ برای کنترل نسخه این پروژه بزرگ یک DVCS غیرآزاد با نام BitKeeper در نظر گرفته شد.

در سال ۲۰۰۵، رابطه تیم توسعه‌دهنده هسته لینوکس و شرکت تجاری توسعه‌دهنده BitKeeper را توسعه داد بر هم خورد، و وضعیت رایگان این ابزار برای پروژه لغو شد. این موضوع تیم توسعه‌دهنده لینوکس (و مخصوصا Linus Torvals، خالق لینوکس) را ترغیب کرد تا ابزار خود را بر اساس برخی از درس‌هایی که در طول استفاده از BitKeeper یاد گرفتند توسعه دهند. برخی از اهداف سیستمی که مد نظر داشتند در زیر آمده است:

- سرعت
- طراحی ساده
- پشتیبانی قوی برای توسعه‌ی غیر-خطی (هزاران branch موازی)
- کاملا توزیع شده
- برآمدن از پس پروژه‌های بزرگ مثل هسته لینوکس (به لحاظ سرعت و همچنین حجم داده‌ها)

از زمان تولد Git در سال ۲۰۰۵، طوری تکامل یافت و رشد کرد تا استفاده از آن ساده باشد و جالب اینجاست که سادگی را به عنوان یک سنگ بنا هنوز حفظ نموده است. Git به طور باور نکردنی سریع است لذا برای پروژه‌های بزرگ بسیار کارآمد است. به علاوه مجهز به یک سیستم branching بسیار قدرتمند برای توسعه غیرخطی است. (فصل ۳ را مطالعه فرمایید).

---

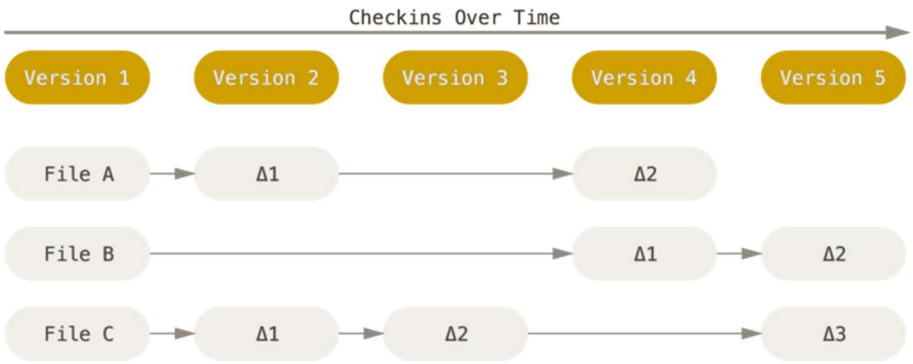
[۲۰] پدیده Creative Destruction به معنی پدیده‌ای است که در آن بسط سریع یک اقتصاد، نیروهایی را به طور خودکار به کار می‌اندازد که به این حالت پایان می‌بخشد و سپس دوره‌ی انقباض فرا می‌رسد. در اینجا یعنی مشقت زیاد جهت نگهداری نسخه‌ها باعث بروز خلاقیت گشت که سرانجام آن تولید سیستم Git شد.

مبانی Git

Git چیست؟ این بخش بسیار مهم است. اگر شما مبانی چگونگی کار با Git را خوب فرا گیرید، آنگاه استفاده موثر از گیت برای شما خیلی ساده‌تر خواهد بود. همان طور که گیت را یاد میگیرید، سعی کنید تا ذهن خود را از چیزهایی که ممکن است در مورد VCSهای دیگر، مثل Subversion و Perforce بدانید پاک کنید. چنین کاری به شما کمک خواهد کرد تا از سردرگمی در زمان استفاده از Git در امان بمانید. نوع ذخیره‌سازی فایل‌ها و نوع نگرش Git به اطلاعات نسبت به سیستم‌های دیگر بسیار متفاوت است. اگرچه رابط کاربری این سیستم‌ها شبیه یکدیگرند.

Snapshot ها و نه تفاوت‌ها

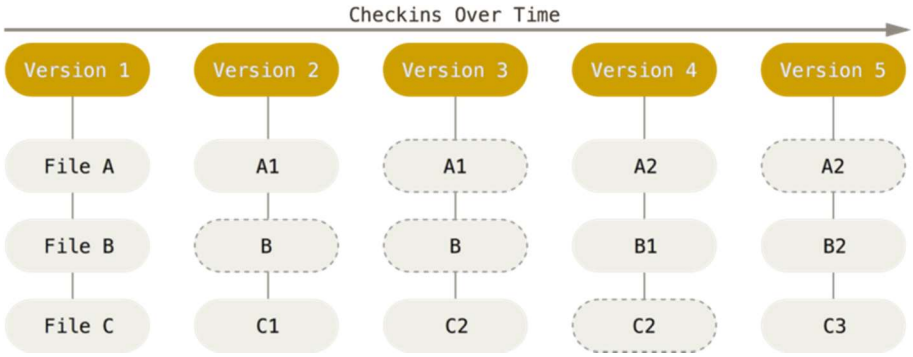
تفاوت اصلی بین گیت و هر VCS دیگر (Subversion و دوستان) نوع نگرش و تفکر گیت در مورد داده‌های خود است. از نظر مفهومی، اکثر سیستم‌های دیگر اطلاعات را به صورت یک لیست از تغییرات مبتنی بر فایل ذخیره می‌کنند. این سیستم‌ها (CVS، Subversion، Perforce، Bazaar و غیره) به اطلاعاتی که خود ذخیره می‌کنند به صورت یک مجموعه از فایل‌ها و تغییرات انجام شده بر روی هر فایل در طول زمان می‌نگرند.



شکل ۴- اطلاعات مربوط به تغییرات تک تک فایل‌ها را نسبت به نسخه اولیه آنها ذخیره می‌کند

گیت به این طریق به داده‌های خود فکر و یا آنها را ذخیره نمی‌کند. در عوض، گیت بیشتر مثل یک مجموعه از snapshot ها به داده‌های خود فکر می‌کند. هر زمانی که شما

commit می‌کنید، یا وضعیت پروژه خود را در گیت ذخیره می‌کنید، اساسا یک snapshot از فایل‌های شما در آن لحظه میگیرد و یک آدرس نسبت به آن snapshot ذخیره میکند. گیت در این فرآیند جهت کارآمدی اگر فایلی تغییر نکرده باشد، فایل را مجددا ذخیره نمی‌کند، فقط به فایل همسان قبلی که قبلا ذخیره شده لینک می‌دهد. گیت در مورد داده‌های خود بیشتر مثل یک جریان از snapshot ها فکر می‌کند.



شکل ۵- snapshot هایی از پروژه در طول زمان را ذخیره می‌کند

این یک تمایز مهم بین گیت و تقریبا همه VCSهای دیگر است. این امر باعث موجب می‌شود تا Git تقریبا در همه جنبه‌های کنترل نسخه که اکثر سیستم‌های دیگر از نسل قبلی کپی کرده بودند تجدید نظر کند. این امر Git را بیشتر شبیه یک file system مینیاتوری و کوچک کرده که یک ابزار بسیار قدرتمند بر روی آن سوار شده است تا یک ابزار VCS ساده. در فصل ۳ برخی از مزایای این نوع نگرش نسبت به داده‌ها را از جمله branching، پوشش خواهیم داد.

تقریبا تمام عملیات به صورت محلی انجام می‌شوند

اکثر عملیات در Git تنها به فایل‌ها و منابع محلی برای اجرا نیاز دارند. عموما هیچ اطلاعاتی از کامپیوتر دیگر بر روی شبکه شما مورد نیاز نیست. اگر شما به CVS ها عادت کردید که اکثر عملیات آن سربار تاخیر شبکه را دارند، این جنبه از Git موجب می‌شود به این مسئله فکر کنید که خدایان سرعت، Git را متبرک به قدرت‌های معنوی کردند.

چون شما تمام تاریخچه‌ی پروژه را درست بر روی دیسک محلی خود دارید، اکثر عملیات تقریبا آنی و بی‌درنگ به نظر می‌رسند.

به طور مثال، برای مرور تاریخچه‌ی پروژه، Git نیاز ندارد تا به سرور متصل شود تا تاریخچه را به دست آورد و آن را برای شما نمایش دهد. Git آن را مستقیما از پایگاه داده‌ی محلی شما می‌خواند. یعنی شما تاریخچه‌ی پروژه را تقریبا به صورت بی درنگ می‌بینید. اگر شما می‌خواهید تغییرات بین نسخه‌ی فعلی یک فایل و نسخه‌ی یک ماه پیش آن را ببینید، Git می‌تواند نسخه یک ماه پیش آن را پیدا کند و یک محاسبه‌ی محلی انجام دهد، به جای این که از سرور بخواهد این کار را انجام دهد یا نسخه‌ی قدیمی‌تر را از سرور بگیرد تا آن را به صورت محلی انجام دهد.

در واقع عملیات بسیار اندکی وجود دارد که شما بدون اتصال به اینترنت یا VPN نتوانید آن را انجام دهید. اگر شما با قطار یا هواپیما در حال سفر هستید و هوس کار کردن به سرتان زده است نباید اصلا نگران commit کردن تغییراتتان باشید. تمام commit هایی که شما انجام داده‌اید در اولین فرصتی که اینترنت یا VPN در دسترس باشد قابل انتقال به سرور اصلی است. در بسیاری از سیستم‌های دیگر انجام این کار یا غیرممکن است یا بسیار آزاردهنده. به طور مثال در Perforce وقتی به سرور متصل نیستید کار زیادی از کنترل نسخه شما ساخته نیست. در این شرایط Subversion و CVS به شما امکان ویرایش فایل‌ها را می‌دهند اما این تغییرات را نمی‌توانید در پایگاه داده‌ی خود commit کنید (زیرا پایگاه داده شما offline است). در ظاهر شاید این تفاوت چندان معضل بزرگی به حساب نیاید، اما در عمل طعم تلخ تجربه آن همیشه در خاطرتان خواهد ماند.

صحت اطلاعات در Git

هر چیزی که بخواهد در Git ذخیره شود، ابتدا checksum [۲۱] آن محاسبه می‌شود و سپس به وسیله همین checksum ارجاع داده می‌شود. چنین عملی موجب می‌شود که

[۲۱] Checksum در واقع یک رشته بلند از کاراکترهاست که به عنوان اثرانگشت یک فایل خاص جهت بررسی نمودن صحت آن فایل در فرآیند انتقال استفاده می‌شود.

در صورت ایجاد کوچکترین تغییری در محتویات فایل یا پوشه‌ای، Git از آن آگاهی پیدا کند. بدین دلیل است که اگر داده‌ای در حین انتقال از دست برود و یا فایلی مخدوش شود، Git سریعا از آن اطلاع پیدا می‌کند.

Git برای تولید checksum از SHA-1 [۲۲] استفاده می‌کند. خروجی تابع SHA-1 یک hash خواهد بود.

این hash یک رشته ۴۰ کاراکتری از کاراکترهای مبنای شانزده است (a-f , 0-9) که از روی محتویات فایل و یا ساختار پوشه مورد نظر در Git محاسبه می‌گردد. در ادامه یک نمونه از hash تولید شده با استفاده از تابع SHA-1 آورده شده است:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

شما با این مقادیر hash در Git بسیار سر و کار خواهید داشت. در حقیقت Git تمامی فایل‌ها را با مقدار hash آن می‌شناسد و اطلاعات را در پایگاه داده‌ی خود بر اساس همین مقدار ذخیره می‌کند نه بر اساس نام فایل‌ها.

Git عموما فقط اطلاعات را اضافه می‌کند

هرگاه عملی در Git انجام می‌پذیرد، تقریباً در تمامی موارد Git داده‌ای به داده‌های خود در پایگاه داده اضافه می‌کند. انجام دادن عملی که برگشت‌پذیر نباشد یا باعث حذف داده‌ای از سیستم شود در این سیستم بسیار سخت است. مشابه اکثر VCS ها، فرد می‌تواند تا قبل از commit هرگونه تغییراتی را انجام دهد؛ ولی به محض commit یک snapshot در Git، امکان حذف آن بسیار سخت است، مخصوصاً اگر شخص، commit خود را به یک repository دیگر push کند. [۲۳]

بهترین روش برای آموختن ابزاری مانند Git سعی و خطا کردن و آزمایش دستورات مختلف آن است. با در نظر گرفتن ویژگی فوق ابدا نباید نگران از دست دادن یا مخدوش شدن اطلاعات خود باشید. به راحتی هر چه تمام‌تر می‌توانید هر آزمایشی که می‌خواهید روی Git انجام دهید. برای یک بررسی دقیقتر و عمیق‌تر در اینکه چگونه گیت داده‌های

---

[۲۲] تابع درهم‌سازی در مقوله‌ی رمزنگاری است. (Secure Hash Algorithm 1)

[۲۳] با دستورات commit و push در فصل آینده (فصل ۲) آشنا خواهید شد.

خود را ذخیره میکند یا چگونه میتوانید داده‌هایی که ظاهرا گم شدند را بازیابی کنید مراجعه نمایید به فصل ۲ بخش چهارم، «لغو تغییرات». [۱]

وضعیت‌های سه‌گانه Git

لطفا توجه فرمایید، درک این بخش کوچک از کتاب تاثیر فراوانی در ادامه فرآیند یادگیری شما خواهد داشت. فایل‌ها در Git می‌توانند در ۳ وضعیت مختلف قرار گیرند. این ۳ وضعیت عبارتند از:

- **Committed**

در این وضعیت اطلاعات در پایگاه داده Git و هیچ خطری داده‌ها را تهدید نمی‌کند.

- **Modified**

بدین معنی است که شما فایلی را تغییر داده‌اید اما هنوز آن را در پایگاه داده commit نکرده‌اید.

- **Staged**

وقتی فایلی در وضعیت modified را در در لیست commit بعدی خود قرار می‌دهید.

برابر این سه وضعیت، سه بخش در یک پروژه تحت کنترل Git به وجود می‌آید:

- **Git directory**

جایی است که گیت metadata و object database پروژه را ذخیره می‌کند. این مهمترین بخش از گیت است، و در واقع همان چیزی است که وقتی شما یک repository را از کامپیوتر دیگر clone می‌کنید ایجاد می‌شود.

- **Working directory**

یک کپی از پروژه، برابر با نسخه انتخابی شماست. فایل‌های این نسخه از پروژه، از پایگاه داده فشرده شده در Git directory بیرون کشیده می‌شود و بر روی دیسک آماده تغییرات و استفاده شما قرار می‌گیرد. [۲]

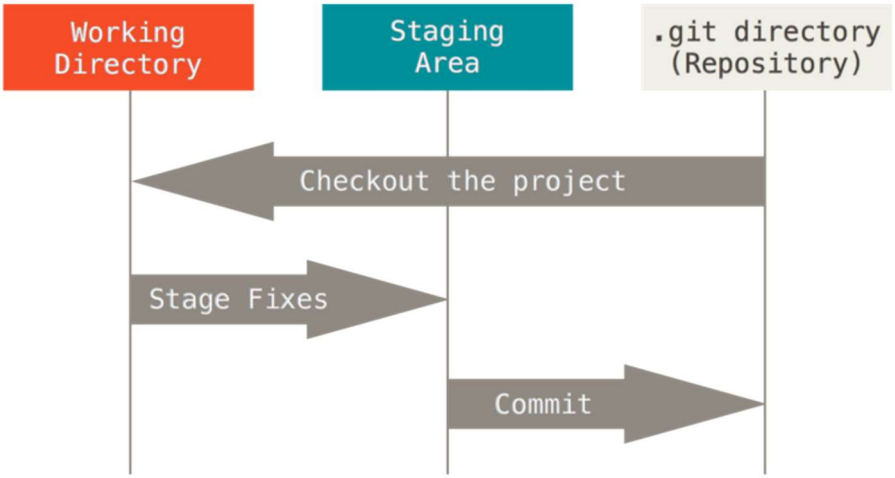
---

[۱] Undoing changes

[۲] به فرآیند بیرون کشیدن نسخه‌ای از یک فایل از درون پایگاه داده فشرده شده که در Git directory قرار دارد، check out می‌گوییم. اصطلاحا می‌گوییم آن فایل را check out کرده‌ایم. همان طور که توضیح داده شد

- **Staging area**

یک فایل است که اکثر مواقع در Git directory قرار می‌گیرد و لیست فایل‌های commit بعدی شما را در خود نگاه می‌دارد. گاهی اوقات تحت عنوان index از این فایل یاد می‌شود اما همان نام staging area رایج‌تر است.



شکل ۶- وضعیت سه‌گانه Git

در زیر یکی از مقدماتی‌ترین workflow های Git را خواهید دید:

۱. فایل‌ها را در working directory خود تغییر می‌دهید.
۲. فایل‌های تغییر یافته را به staging area اضافه می‌کنید.
۳. یک commit را انجام می‌دهید، که فایل‌ها را همان طور که در staging area هستند می‌گیرد و آن را به صورت دائمی در Git directory پروژه شما ذخیره میکند.

اگر یک نسخه خاص از یک فایل در Git directory قرار دارد، آن فایل commit شده تلقی می‌شود. اگر فایلی تغییر کرده است و به staging area اضافه شده است آن فایل

---

check out کردن، یک نسخه خاص از یک فایل را از Git directory به روی دیسک یا در واقع همان working directory می‌آورد.

اصطلاحا staged شده است. اگر فایلی check out شده و بعد از آن تغییر کرده است اما هنوز staged نشده است آن فایل در وضعیت modified قرار دارد.

در فصل ۲ در مورد این وضعیت‌ها و این که چطور می‌توانید از آن‌ها بهره‌مند شوید بیشتر خواهید آموخت.

Command Line

روش‌های مختلف زیادی برای استفاده از Git وجود دارند. Command line و رابط‌های کاربری گرافیکی (GUI) زیادی با قابلیت‌های متفاوت وجود دارند. در این کتاب، ما از command line استفاده خواهیم کرد. به این جهت که command line در دسترس همگان قرار دارد و وابسته به platform خاصی نیست و تمامی دستورات Git را می‌توان با آن اجرا کرد.

اکثر GUIها جهت سادگی تنها برخی از قابلیت‌های Git را پیاده‌سازی می‌کنند و انتخاب آن کاملا بر اساس سلیقه است. به علاوه اگر شما بتوانید با command line کار کنید بالطبع می‌توانید با تمامی GUIها نیز کار کنید در حالی که عکس این موضوع صدق نخواهد کرد.

بنابراین انتظار می‌رود که بدانید چگونه با Terminal در Mac و Command Prompt یا Powershell در ویندوز کار کنید. اگر اولین بار است که این اسامی را مشاهده می‌کنید بهتر از خواندن این کتاب را در اینجا متوقف نمایید و بعد از آموختن یکی از ابزارهای بالا به آموختن Git ادامه دهید.

نصب Git

قبل از اینکه شما استفاده از گیت را شروع کنید، باید آن را بر روی کامپیوتر خود داشته باشید. حتی اگر از قبل نصب شده است، شاید بهتر باشد آن را به آخرین نسخه به روز رسانی کنید. برای نصب Git چند راه وجود دارد. آسان‌ترین آن این است که نسخه قابل نصب آن را دانلود و سپس نصب نمایید. راه پیچیده‌تر آن است که کد Git را دریافت و سپس آن را compile نمایید.

این کتاب بر اساس نسخه Git ۲,۰,۰ نوشته شده است. اگرچه اکثر command هایی که ما استفاده می‌کنیم حتی در نسخه‌های قدیمی‌تر نیز کار می‌کنند، با این حال ممکن است تعداد کمی از command ها بر روی نسخه‌های قدیمی‌تر کار نکنند یا رفتارشان کمی متفاوت باشد.

اما در مورد نسخه‌های جدیدتر، از آنجایی که Git به شدت backward compatible[۱] است، نسخه‌های بعدی باید کاملا درست کار کنند.

نصب بر روی Linux

اگر از Fedora استفاده می‌کنید، می‌توانید Git را توسط yum نصب نمایید.

```
$ sudo yum install git-all
```

اگر با distribution های مبتنی بر Debian مانند Ubuntu کار می‌کنید، apt-get را امتحان کنید:

```
$ sudo apt-get install git-all
```

برای گزینه‌های بیشتر، دستورالعمل‌هایی برای نصب بر روی چندین flavor مختلف Unix بر روی وب سایت Git، در ***http://git-scm.com/download/linux*** وجود دارد.

نصب بر روی Mac

چندین روش برای نصب Git بر روی Mac وجود دارند. اگر XCode یا Command

---

[۱] در این جا به معنی آن است که با آمدن نسخه‌های جدیدتر، command های قدیمی، همچنان قابل استفاده هستند.

Line Tools آن را بر روی Mac نصب کرده‌اید Git نیز بر روی سیستم شما نصب شده است. برای این که مطمئن شوید، terminal را باز کنید و command زیر را اجرا کنید:

```
$ git --version
git version 2.10.1 (Apple Git-78)
```

راه حل دیگر برای نصب استفاده از Homebrew است. برای نصب از این روش می‌توانید command زیر را در terminal اجرا نمایید:

```
$ brew install git
```

همچنین می‌توانید به راحتی نسخه نصبی Git مخصوص سیستم عامل Mac را از لینک زیر دریافت نمایید و به راحتی آن را نصب کنید.

http://git-scm.com/download/mac

شکل ۷- نصب Git بر روی سیستم عامل Mac

نصب بر روی Windows

برای Windows نیز چندین روش نصب وجود دارد. معتبرترین آن‌ها این است که به آدرس زیر بروید و بعد به صورت خودکار دانلود Git آغاز می‌شود.

http://git-scm.com/download/win

توجه فرمایید پروژه‌ای که دانلود می‌کنید Git for Windows نام دارد از خود پروژه Git جداست. برای اطلاعات بیشتر در مورد آن می‌توانید به آدرس زیر مراجعه نمایید.

https://git-for-windows.github.io

روش ساده‌ی دیگر نصب GitHub for Windows است. بعد از نصب این برنامه Git command line و همچنین یک رابط کاربری گرافیکی برای استفاده از Git نصب خواهد شد. برای اطلاعات بیشتر می‌توانید به آدرس زیر مراجعه فرمایید:

http://windows.github.com

اولین گام بعد از نصب Git

اکنون شما گیت را بر روی سیستم خود دارید. در اولین قدم بعد از نصب لازم است محیط Git را برای خود تنظیم نمایید. این کار را فقط یک بار لازم است تا انجام دهید. البته هر زمانی که بخواهید می‌توانید تنظیمات را تغییر دهید.

همراه با Git ابزاری ارائه می‌شود تحت عنوان git config. همان طور که از نامش پیداست این ابزار در واقع تمام متغیرهای قابل config را در اختیار شما می‌گذارد تا مقدادیر آن‌ها را بخوانید یا تغییر دهید. این متغیرها کنترل‌کننده تمامی جنبه‌های ظاهری و رفتاری Git می‌شوند.

این متغیرها در ۳ مکان مختلف نگهداری می‌شوند:

/etc/configfile

مقادیری که برای تمامی کاربران سیستم و تمامی repository های آنها نگهداری می‌شود. اگر شما system-- را به عنوان option به git config پاس دهید، آن گاه تنظیمات از این فایل خوانده می‌شوند و تغییرات بر روی این فایل اعمال خواهد شد.

~/.config/git/config یا ~/.gitconfig

مقادیر تنظیمات که مخصوص یک کاربر خاص است در این مکان ذخیره می‌شود. اگر global-- را به عنوان option به git config پاس دهید، تنظیمات از این فایل خوانده و تغییرات بر روی همین فایل اعمال خواهد شد.

فایل config درون git directory (پوشه git/config.) یک repository خاص

مقادیر تنظیمات مربوط به همان repository خاص خواهد شد. به ترتیب هر کدام از این سطوح، تنظیمات سطح قبل را تحت‌الشعاع قرار می‌دهد. بنابراین مقادیر git/config. بر مقادیر etc/configfile/ برتری دارد.

تنظیمات هویت شما

بعد از این که نصب Git با موفقیت انجام شد، اولین و مهمترین کار تنظیم نام کاربری و آدرس ایمیل خودتان است. علت اهمیت این امر آن است که هر commit در Git از این اطلاعات استفاده می‌کند و به طور تغییرناپذیر جزیی از commit هایی را تشکیل می‌دهند

که شما انجام می‌دهید.

```
$ git config --global user.name “Reza Ahmadi”
$ git config --global user.email r.odises@gmail.com
```

تکرار می‌کنیم، اگر شما global-- را به عنوان option به git config پاس دهید، فقط و فقط یک بار لازم است این تنظیمات را انجام دهید. مگر اینکه بخواهید برای یک پروژه خاص این مقادیر global را تغییر دهید. برای این کار همان طور که در قبل توضیح داده شد بایستی فایل config مربوط به آن پروژه خاص را تنظیم نمایید تا مقادیر global تحت‌الشعاع قرار گیرند.

تنظیمات مربوط به Editor شما

اکنون که شناسه شما تنظیم شد، میتوانید editor پیش‌فرض را تنظیم نمایید. Editor زمانی به کار می‌آید که git از شما بخواهد متنی را ویرایش نمایید. اگر editor تنظیم نشده باشد، editor پیش‌فرض سیستم شما استفاده می‌شود.

اگر شما میخواهید از editor دیگری استفاده کنید، مثل Emacs، بایستی مانند زیر عمل کنید:

اگر بر روی Windows از Git استفاده می‌کنید و می‌خواهید از editor دیگری مثل ++Notepad استفاده نمایید می‌توانید به صورت زیر اقدام نمایید:

بر روی سیستم x86

```
$ git config --global core.editor "'C:/Program
Files/Notepad++/notepad++.exe' -multiInst -nosession"
```

بر روی سیستم x64

```
$ git config --global core.editor "'C:/Program Files
(x86)/Notepad++/notepad++.exe' -multiInst -nosession"
```

Vim، Emacs و ++Notepad از جمله مشهورترین editor هایی هستند که اغلب

توسط توسعه‌دهندگان بر روی سیستم‌های مبتنی بر UNIX مثل Linux و OS X یا Windows استفاده می‌شوند. اگر شما با هیچ یک از این editor ها آشنا نیستید، احتمالا باید کمی در موردشان تحقیق کنید تا یکی را به عنوان editor پیش‌فرض Git انتخاب نمایید.

مشاهده تنظیمات

اگر می‌خواهید تنظیمات خود را چک کنید، می‌توانید از فرمان git config --list استفاده کنید:

```
$ git config --list
user.name=Reza Ahmadi
user.email=r.odises@gmail.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

به این نکته توجه فرمایید که ممکن است شما یک key را در لیست تنظیمات خود ببینید. این بدین خاطر است که Git آن key را از فایل‌های config مختلف می‌خواند برای مثال یکی را از etc/gitconfig/ و دیگری را از gitconfig./~.

شما همچنین می‌توانید مقدار یک key خاص را با الگوی git config <key> چک کنید.

```
$ git config user.name
Reza Ahmadi
```

راهنمای Git

اگر شما در حال استفاده از گیت به کمک نیاز دارید، سه روش وجود دارد تا صفحه راهنمای[۱] آن command را ببینید.

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

برای مثال، صفحه راهنمای فرمان config با اجرای دستور زیر بدست خواهد آمد.

```
$ git help config
```

خلاصه

شما باید اطلاعات ابتدایی در مورد این مسئله داشته باشید که Git چیست و چه تفاوت‌هایی با سیستم کنترل نسخه متمرکزی که ممکن است قبلا از آن استفاده کرده باشید دارد. شما همچنین اکنون باید یک نسخه در حال کار از Git بر روی سیستم خود داشته باشید که با هویت شما config شده باشد. اکنون زمان این است که برخی مبانی Git را یاد بگیرید.

[۱] Manual page (manpage)