# **資深軟體工程師引導：新手工程師如何高效上手Git**

作為一名資深軟體工程師，深知版本控制系統對於現代軟體開發的重要性。在眾多版本控制工具中，Git憑藉其分散式特性、強大的分支管理能力以及高效的協作機制，已成為業界標準。對於剛踏入軟體領域的新手工程師而言，紮實掌握Git不僅是基本技能，更是提升開發效率、融入團隊協作的關鍵。本報告將從核心概念出發，結合最常用且實用的操作，引導新手工程師逐步上手Git。

## **I. Git核心概念與工作原理**

### **1.1 版本控制系統概述**

版本控制系統（Version Control System, VCS）的核心目的在於記錄一個或多個文件的內容變化，以便開發者能夠在未來查閱特定版本的修訂情況，並有效管理程式碼 1。它不僅限於專案原始碼，任何類型的文件都可以進行版本控制 1。

VCS為開發者帶來多重優勢：

* **歷史記錄與追溯：** 系統會保留對程式碼庫所做變更的詳細歷史記錄。每一次提交（Commit）都代表專案在特定時間點的快照，包含變更內容、變更者及變更時間 3。這提供了寶貴的背景資訊，使開發者能理解程式碼庫的演變，並在必要時回溯到任何歷史狀態 3。
* **協作與團隊合作：** VCS支援團隊成員間的協作，允許多位開發者同時在同一個專案上工作，並追蹤每個人的貢獻 3。它提供工具來合併變更並解決衝突，確保團隊成員可以獨立工作，然後將各自的更改合併到一個共同的程式碼庫中，從而提高工作效率並減少錯誤 2。
* **分支與實驗：** 分支功能允許開發者創建獨立的開發線，這對於開發新功能、修復錯誤或進行實驗而不影響主程式碼庫的穩定性至關重要 3。
* **回滾與復原：** 若出現錯誤或引入問題，VCS允許開發者回滾到程式碼的先前版本，將程式碼庫恢復到已知的工作狀態 3。
* **可追溯性與問責制：** 系統為程式碼庫的所有變更提供了清晰的稽核軌跡，開發者可以清楚地看到誰在何時、為何進行了特定變更，這有助於識別問題根源、追蹤錯誤並確保團隊成員對其貢獻負責 3。

### **1.2 集中式與分散式版本控制系統**

版本控制系統主要分為兩種類型：集中式版本控制系統（Centralized Version Control System, CVCS）和分散式版本控制系統（Distributed Version Control System, DVCS）。

* **集中式版本控制系統 (CVCS)：**
  + **原理：** 所有的文件和版本都集中儲存在一個中央伺服器上。開發者透過與伺服器互動來獲取最新版本並進行開發 4。代表性系統包括SVN (Subversion) 4。
  + **優點：** 管理相對簡單，所有版本資訊集中儲存，易於管理和控制；安全性較高，團隊成員只能透過伺服器協作，有助於保護程式碼安全 2。
  + **缺點：** 存在單點故障風險，如果中央伺服器發生故障，所有團隊成員將無法正常工作；必須依賴網路連線才能進行版本控制操作，限制了工作的靈活性 1。
* **分散式版本控制系統 (DVCS)：**
  + **原理：** 每個開發者都擁有完整的程式碼倉庫副本，包含完整的歷史記錄。開發者可以在本地進行版本控制操作，不需始終依賴中央伺服器 1。Git、Mercurial是典型的DVCS 4。
  + **優點：**
    - **分散式協作與離線操作：** 每個開發者都擁有完整的程式碼倉庫，可以在本地獨立開發和提交，不受網路狀態影響，極大提升工作靈活性 1。即使中央伺服器故障，也可從任何一個本地倉庫恢復 1。
    - **高效能協作：** 開發者可以在本地創建分支、提交變更並進行合併，無需每次都與中央伺服器互動，這顯著減少了伺服器負載，提高了團隊協作效率 4。
    - **強大的分支管理：** Git在分支管理方面表現卓越，創建、合併和切換分支都非常快速高效，允許團隊成員並行開發不同功能或修復錯誤 1。
    - **高安全性：** 由於每個開發者都有完整的本地程式碼倉庫，即使中央伺服器故障或資料遺失，本地倉庫仍包含完整的專案歷史，可用於恢復和重建，提供了更高的資料可靠性 4。
  + **缺點：** 初始複製倉庫時可能較慢；相對於CVCS，學習曲線較陡峭 4。

Git作為最流行的DVCS之一，其靈活性和高效性使其成為現代開發團隊的首選 4。

### **1.3 Git的資料處理方式：快照而非差異**

Git在保存和處理資料的方式上與其他版本控制系統有顯著差異。大多數傳統VCS（如CVS、Subversion）將資訊儲存為文件變更的列表，它們將儲存的資訊視為一組基礎文件和隨時間累積的差異（增量）。要獲取文件的最終版本，需要將原始文件與所有增量變更相加，這在增量非常多時會消耗時間和效能 1。

相反，Git採用的是**直接記錄快照**的方式 1。每次提交更新或保存專案狀態時，Git主要對當時的全部文件製作一個快照並保存這個快照的索引 1。為了提高效率，如果文件沒有修改，Git不會重新儲存該文件，而是只保留一個連結指向之前儲存的文件 1。Git將資料視為一個

**快照流** 1。這種快照方式使得Git在處理歷史記錄和分支時更加高效和靈活。

### **1.4 Git的三種狀態與工作區域**

理解Git的三種文件狀態及其對應的工作區域，是掌握Git操作的基石 1。

* **三種文件狀態：**
  + **已提交 (committed)：** 資料已安全地儲存在本地Git資料庫中 1。
  + **已修改 (modified)：** 文件已被修改，但尚未保存到Git資料庫中 1。
  + **已暫存 (staged)：** 對一個已修改文件的當前版本做了標記，使其包含在下次提交的快照中 1。
* **三種核心工作區域：**
  + **工作目錄 (Working Directory)：** 這是開發者實際編輯文件的目錄，也是專案文件所在的位置 1。當你在這裡修改文件時，這些文件處於「已修改」狀態。
  + **暫存區 (Staging Area / Index)：** 這是一個輕量級的區域，通常儲存在.git目錄下的index文件（.git/index）中 1。它用於存放你準備在下一次提交中包含的變更快照。文件從工作目錄透過  
    git add命令進入暫存區，此時文件處於「已暫存」狀態 7。
  + **Git 儲存庫 (Git Repository / 版本庫)：** 這是Git用來儲存專案所有版本歷史和元資料的地方，通常是工作目錄中的隱藏.git子目錄 1。當你執行  
    git commit命令時，暫存區的內容會被永久儲存到這裡，形成一個新的提交物件，文件狀態變為「已提交」 1。

### **1.5 Git的核心物件**

Git的內部運作基於三種核心物件，它們共同構成了Git的內容尋址檔案系統 4：

* **提交物件 (Commit Object)：**
  + **定義：** 提交物件是Git中的基本單位，每次提交都會生成一個唯一的提交物件 4。它包含了專案在該時間點的快照資訊以及提交相關的元資料，例如作者、提交時間和提交訊息 4。
  + **工作原理：** 當你提交變更時，Git會創建一個提交物件，其中包含專案文件的快照，記錄了這次提交的作者、時間、提交訊息，以及指向前一個提交物件的引用 4。這樣就形成了一條提交歷史鏈，可以方便地回溯專案的歷史狀態 4。每個提交物件都由一個40字元的SHA-1雜湊值唯一標識 8。
* **樹物件 (Tree Object)：**
  + **定義：** 樹物件是目錄結構的快照，它對應著專案文件在某個提交時的狀態 4。每次提交都會生成一個樹物件，其中包含了當前專案文件和目錄的結構和內容資訊 4。
  + **工作原理：** 當你提交變更時，Git會創建一個樹物件，該物件對應著當前專案文件和目錄的結構，記錄了文件的位置、名稱和內容資訊 4。透過樹物件，可以快速查找和恢復專案文件在某個提交時的狀態 4。樹物件類似於UNIX文件系統中的目錄，它包含指向blob物件（文件內容）或其他樹物件（子目錄）的SHA-1指針 8。
* **引用 (Reference)：**
  + **定義：** 引用是指向提交物件的指針，它可以是分支、標籤或其他指向提交的指針 4。引用幫助我們標記提交歷史中的重要點，方便查找和管理 4。
  + **工作原理：** 當你創建一個分支或標籤時，Git會創建一個引用，它指向某個提交物件 4。隨著新的提交產生，引用也會隨之移動，始終指向最新的提交 4。透過引用，你可以方便地找到不同分支或標籤對應的提交歷史 4。

理解這些核心物件及其相互關係，有助於新手工程師更深入地理解Git如何追蹤文件變化、保存歷史記錄並實現版本控制，為在實際專案中靈活高效地使用Git奠定基礎。

## **II. Git基礎操作與實踐**

本節將介紹Git最常用且實用的基本操作，這是新手工程師日常開發中必不可少的技能。

### **2.1 安裝與初始設定**

在開始使用Git之前，首先需要在你的開發環境中安裝Git並進行基本配置。

* **安裝Git：**
  + **Mac：** 最簡單的方式是安裝Xcode Command Line Tools。在Terminal中首次嘗試運行git命令，系統會提示你安裝。若需更新版本，可使用二進位安裝程式 10。
  + **Windows：** 可從Git官方網站下載安裝程式（git-scm.com/download/win） 10。另一個簡單方法是安裝GitHub for Windows，它包含圖形化和命令行版本的Git，並支援Powershell 10。
* **初始設定（配置用戶名和郵箱）：**
  + Git的每次提交都會記錄作者資訊，因此配置你的用戶名和郵箱是使用Git的第一步，也是至關重要的一步 4。
  + **命令：**  
    Bash  
    git config --global user.name "你的名字"  
    git config --global user.email "你的郵箱@example.com"
  + **重要性：** 這些配置會儲存在你的全局配置文件（~/.gitconfig 或 ~/.config/git/config）中，對所有Git倉庫生效 10。在團隊協作中，每個提交都能明確追溯到具體的開發者，這建立了程式碼變更的  
    **可追溯性和問責制**。當出現問題時，可以迅速定位到相關的提交和負責人，促進團隊內部溝通和問題解決效率，這是良好協作的基礎。

### **2.2 初始化本地儲存庫：git init**

當你打算使用Git來管理一個現有的專案，或從頭開始一個新專案時，第一步是初始化一個Git儲存庫。

* **命令：**  
  Bash  
  cd /path/to/your/project # 導航到你的專案目錄  
  git init # 初始化一個新的 Git 儲存庫
* **效果：** 執行git init命令後，Git會在該目錄中創建一個名為.git的隱藏子目錄 10。這個  
  .git目錄是Git儲存庫的骨幹，包含了所有Git版本控制所需的必要文件和數據 10。除了這個隱藏目錄，專案的其他部分不會有任何改變 11。
* **驗證：** 成功運行後，你可以使用ls -a命令來確認是否成功創建了.git目錄 11。

### **2.3 基本工作流程：修改、暫存、提交**

這是Git最核心且頻繁使用的操作循環，理解這個流程是掌握Git的關鍵 1。

* **修改 (Modify)：** 在你的工作目錄中對文件進行更改、新增或刪除。此時文件處於「已修改」狀態。
* **暫存 (Stage)：** 當你完成一部分修改，並準備將這些修改納入下一次提交時，你需要使用git add命令將這些文件的快照放入暫存區。此時文件處於「已暫存」狀態。
* **提交 (Commit)：** 一旦暫存區中的內容準備就緒，你就可以使用git commit命令，將暫存區的快照永久性地儲存到本地Git儲存庫中，形成一個新的版本記錄。此時文件處於「已提交」狀態。

### **2.4 追蹤檔案：git add**

git add命令的目的是將工作目錄中新創建或修改的文件添加到暫存區，使其被Git追蹤，並納入下一次提交的快照中 7。

* **命令：**
  + git add <file\_name>：將指定的文件添加到暫存區 11。
  + git add.：將當前目錄下所有新文件和修改過的文件（不包括被忽略的文件）添加到暫存區 11。
* **狀態變化：** 運行git add後，文件的狀態會從"Untracked files" (未追蹤) 或 "Modified" (已修改但未暫存) 變為 "Changes to be committed" (已暫存，準備提交) 12。

### **2.5 查看狀態：git status**

git status命令是你在使用Git時最常用也最有用的命令之一，它能清晰地顯示你工作目錄和暫存區的當前狀態 10。

* **命令：** git status 11。
* **可查看內容：**
  + 你當前所在的本地分支資訊 14。
  + 未追蹤的文件 (Untracked files)：這些是你在工作目錄中新增但尚未被Git追蹤的文件 12。
  + 已修改但未暫存的文件：這些是已被Git追蹤但你在工作目錄中修改後尚未git add的文件。
  + 已暫存但未提交的文件 (Changes to be committed)：這些是已使用git add命令添加到暫存區，準備在下一次git commit中提交的文件 12。
  + 如何將文件從暫存區移除或撤銷工作目錄修改的提示 14。
* **狀態碼簡要解釋：** 在某些Git輸出中，你可能會看到簡潔的狀態碼，例如：M (Modified - 已修改), A (Added - 已新增), D (Deleted - 已刪除), R (Renamed - 已重命名), C (Copied - 已複製), U (Updated but unmerged - 已更新但未合併) 13。

對於新手工程師而言，git status是你的「指南針」和「安全網」。新手工程師在初學Git時，常常會對Git的「三區」狀態感到迷茫，不確定哪些文件已修改、哪些已暫存。git status命令提供了清晰的當前狀態概覽，包括未追蹤、已修改、已暫存的文件，甚至會提供下一步操作的提示。這極大地幫助新手在執行git add或git commit前確認操作的範圍，避免誤操作。培養頻繁使用git status的習慣，能顯著降低操作失誤的風險，提高對Git工作流程的掌控感，是新手快速上手的關鍵。

### **2.6 提交變更：git commit**

git commit命令用於將暫存區的內容作為一個新的快照永久儲存到本地儲存庫中。每次提交都會創建一個新的提交物件，代表專案的一個穩定版本 7。

* **命令：**  
  Bash  
  git commit -m "你的提交訊息"  
  + git commit -m "你的提交訊息"：使用-m選項可以直接在命令行中提供簡潔的提交訊息 11。
  + 如果省略-m選項，Git會打開你配置的預設文本編輯器（如Vim或Nano），讓你輸入更詳細的提交訊息。通常，提交訊息的第一行是簡潔的標題，後面空一行，然後是詳細的正文 15。
* **提交訊息規範：** 一條完整的提交訊息通常包含Header和Body，其中Header是必須的，Body是可選的 15。Header應簡潔扼要（通常不超過50個字元），並由  
  type、scope和subject三部分組成，例如：feat(login): add user authentication 15。常見的  
  type包括feat（新增功能）、fix（修復錯誤）、docs（文件）、refactor（程式碼重構）、chore（維護）等 15。
* **重要性：** 每次提交都應附帶清晰、有意義的提交訊息。這不僅有助於你未來回顧自己的工作，也極大地幫助團隊成員理解你的變更，是良好程式碼習慣的體現 15。適度的提交粒度也很重要，一次提交只應包含一個相關的改動，例如解決一個bug或增加一個新功能，避免將多個不相關的改動混在一起提交 17。

### **2.7 查看提交歷史：git log**

git log命令用於顯示專案的提交歷史記錄，讓你能夠追溯專案的演變過程 8。

* **命令：** git log 12。
* **可查看內容：** 每個提交的唯一SHA-1雜湊值（提交ID）、作者資訊、提交日期和時間、以及提交訊息 12。
* **常用選項：**
  + git log --oneline：以一行簡潔的方式顯示每個提交，只顯示提交ID的前幾位和提交訊息的第一行。
  + git log --graph：以ASCII圖形方式顯示分支和合併歷史，視覺化地呈現提交之間的關係。
  + git log -p：顯示每次提交的詳細變更內容（即diff），讓你看到每個文件具體修改了什麼。
  + git log --author="你的名字"：按作者過濾提交記錄。
  + git log --since="2 weeks ago"：查看最近兩週的提交。

## **III. Git分支管理：團隊協作的利器**

### **3.1 分支的意義與重要性**

分支是Git最強大也最受歡迎的功能之一，它允許開發者從專案歷史中的某個特定提交分支出來，創建一個獨立的工作副本或開發線 3。

* **核心意義：** 實現並行開發。它使多個開發者或同一個開發者能夠同時在不同的功能、錯誤修復或實驗上工作，而不會相互干擾主開發線的穩定性 3。
* **重要性：**
  + **隔離開發：** 新功能、錯誤修復或實驗可以在獨立的分支上進行，避免對主程式碼庫造成不穩定或破壞性的影響 3。
  + **靈活實驗：** 分支鼓勵開發者大膽嘗試新想法、探索不同的實現方案，即使實驗失敗，也不會污染主分支，可以輕鬆丟棄或重新開始 3。
  + **團隊協作基石：** 在現代軟體開發中，團隊成員通常在各自的特性分支上工作，完成後再將這些分支合併回主分支。這種模式是高效團隊協作的基礎 5。

### **3.2 建立與切換分支：git branch, git checkout, git switch**

* **建立分支：**
  + git branch <branch\_name>：這個命令會在本地儲存庫中創建一個新的分支指針，但不會自動將你的工作目錄切換到這個新分支 20。
  + git checkout -b <new\_branch\_name>：這是一個常用的快捷命令，它會創建一個新的分支，並立即將你的工作目錄切換到這個新分支上 20。
  + git switch -c <new\_branch\_name> (Git 2.23+ 推薦)：這是Git 2.23版本後引入的新命令，語義更清晰，專門用於創建並切換到新分支。
* **切換分支：**
  + git checkout <branch\_name>：將你的工作目錄切換到指定的現有分支。當你切換分支時，Git會自動重置你的工作目錄，使其看起來像回到了你在那個分支上最後一次提交的樣子。Git會自動添加、刪除、修改文件以確保此時你的工作目錄與該分支最後一次提交時的樣子一模一樣 1。
  + git switch <branch\_name> (Git 2.23+ 推薦)：這是git checkout的替代命令，專門用於切換分支。
* **查看分支：**
  + git branch：列出所有本地分支，當前分支會以星號標示 20。
  + git branch -r：列出所有遠端追蹤分支 22。
  + git branch -a：列出所有本地和遠端追蹤分支 22。

### **3.3 合併分支：git merge**

合併是將不同分支上的變更整合到一起的過程 4。

* **命令：** git merge <source\_branch>：將指定來源分支的變更合併到當前所在的分支 21。
* **合併類型：**
  + **快進合併 (Fast-forward Merge)：** 如果目標分支自從創建來源分支後沒有新的提交，Git會直接將目標分支的指針「快進」到來源分支的最新提交，不會產生新的合併提交。
  + **三向合併 (Three-way Merge / No-fast-forward Merge)：** 如果目標分支和來源分支都有各自獨立的提交歷史，Git會找到它們最近的共同祖先，然後將兩個分支的變更與共同祖先進行比較，生成一個新的合併提交。使用--no-ff選項可以強制生成合併提交，即使可以快進合併，這有助於保留完整的歷史記錄 22。
* **合併衝突：** 當兩個分支修改了同一個文件的同一部分，或者一個分支刪除了文件而另一個分支修改了它時，就會發生合併衝突 23。Git無法自動解決這些衝突，需要開發者手動介入 23。

### **3.4 分支管理策略簡介**

在團隊協作中，選擇合適的分支策略至關重要，它能規範開發流程，提高效率並減少混亂 20。

* **Git Flow：**
  + **概念：** 一種較為成熟和規範的分支管理流程，定義了長期存在的master（或main）和develop分支，以及短期存在的feature（功能）、release（發布）、hotfix（熱修復）分支 20。
  + **適用場景：** 適合需要嚴格發布管理、多版本並存或有複雜發布流程的專案 26。
  + **優點：** 流程清晰，責任明確，有助於大型團隊和複雜專案的管理 24。
  + **缺點：** 學習曲線較陡峭，分支數量較多，對於小型或快速迭代的專案可能顯得過於繁瑣 26。
* **GitHub Flow：**
  + **概念：** 一種更輕量級、簡化的分支策略，通常只有一個長期存在的main（或master）分支。所有新功能或錯誤修復都從main分支創建feature分支，完成後透過Pull Request（PR）審核並合併回main 24。
  + **適用場景：** 適合小型、溝通能力強的開發團隊，以及追求持續交付的專案 26。
  + **優點：** 靈活輕便，易於理解和採用，支持快速迭代和持續部署，有助於快速反饋和持續改進 26。
  + **缺點：** 對於具有複雜程式碼庫和多個長期功能分支的大型專案適用性有限，且缺乏正式的發布結構 26。

對於新手工程師，建議從GitHub Flow開始，因為其簡單直觀的流程能更快地融入團隊協作。隨著經驗的增長，再逐步理解Git Flow等更複雜的策略。

## **IV. Git遠端儲存庫互動**

在團隊協作中，與遠端儲存庫的互動是不可或缺的一環。這包括將遠端倉庫複製到本地、同步遠端變更以及將本地變更推送到遠端。

### **4.1 複製遠端儲存庫：git clone**

git clone命令用於將遠端Git儲存庫複製到本地電腦。這會創建一個完整的本地儲存庫副本，包括所有文件、歷史記錄和分支 4。

* **命令：** git clone <repository\_URL>
* **效果：** 執行此命令後，Git會自動在本地創建一個與遠端倉庫同名的目錄，並在其中初始化一個本地Git儲存庫（.git目錄），同時將遠端倉庫的內容拉取到本地工作目錄中，並自動建立本地master（或main）分支與遠端origin/master（或origin/main）分支的追蹤關係 30。

### **4.2 同步遠端變更：git fetch 與 git pull**

* **git fetch：**
  + **命令：** git fetch [remote-name] (通常是origin)
  + **作用：** 這個命令會連接到遠端專案，並從遠端專案中將你還沒有的資料全部下載到你的本地儲存庫中 30。它只會下載資料到本地倉庫，並更新遠端追蹤分支（例如  
    origin/master），但**不會自動合併**你的任何工作內容，也不會修改你當前的工作目錄 30。
  + **使用場景：** 當你想查看遠端倉庫的最新狀態，但又不想立即將這些變更應用到你的本地分支或工作目錄時，git fetch非常有用。
* **git pull：**
  + **命令：** git pull [remote-name][branch-name] (通常簡寫為git pull)
  + **作用：** git pull實際上是git fetch和git merge兩個命令的組合 30。它會從遠端儲存庫獲取最新變更，然後自動將這些變更合併到你當前所在的本地分支 30。
  + **使用場景：** 當你想要將遠端倉庫的最新變更同步到你的本地分支並立即應用時，git pull是首選。在開始開發前，建議先執行git pull以確保從最新版本開始工作，減少潛在的合併衝突 31。

### **4.3 推送本地變更：git push**

git push命令用於將你的本地儲存庫中的變更（提交）上傳到遠端儲存庫 30。

* **命令：** git push [remote-name][branch-name] (通常是git push origin <your\_branch\_name>)
* **作用：** 它會將你當前本地分支的所有相關提交物件推送到指定的遠端儲存庫 30。
* **注意事項：** git push只有在你對遠端伺服器有寫入權限，並且在你推送之前沒有其他人推送到該分支時才會成功 32。如果其他人在你之前推送了變更，你的推送將會被拒絕。此時，你必須先  
  git pull拉取他們的變更，解決任何潛在的衝突，然後再嘗試git push 32。第一次推送本地新分支到遠端時，通常需要使用  
  -u（或--set-upstream）選項來建立本地分支與遠端分支的追蹤關係，例如：git push -u origin main 11。

### **4.4 遠端協作流程**

典型的遠端協作流程通常涉及以下步驟 31：

1. **克隆專案：** git clone <remote\_repository\_URL> 將遠端倉庫複製到本地。
2. **創建特性分支：** 從主分支（如main或develop）創建一個新的特性分支，用於開發新功能或修復bug：git checkout -b feature/your-feature-name 20。
3. **在分支上開發：** 在你的特性分支上進行程式碼修改、新增、刪除等操作。
4. **頻繁提交：** 定期將你的工作提交到本地分支：git add. 和 git commit -m "commit message"。
5. **同步遠端變更：** 在將本地變更推送到遠端之前，先拉取遠端主分支的最新變更到你的特性分支，以減少衝突：git pull origin main (如果發生衝突，需要先解決)。
6. **推送到遠端：** 將你的特性分支推送到遠端倉庫：git push -u origin feature/your-feature-name。
7. **發起Pull Request (PR)：** 在托管平台（如GitHub、GitLab）上，向主分支發起Pull Request。PR提供了一個程式碼審查機制，其他團隊成員可以查看你的變更、提供評論和建議 5。
8. **程式碼審查與合併：** 團隊成員對PR進行審查，確保程式碼品質和一致性。審查通過後，由專案維護者將特性分支合併到主分支。
9. **刪除分支（可選）：** 功能合併後，可以刪除本地和遠端的功能分支：git branch -d feature/your-feature-name 和 git push origin --delete feature/your-feature-name 20。

## **V. Git衝突解決與撤銷操作**

即使遵循良好的協作流程，衝突和操作失誤仍可能發生。掌握如何解決衝突和撤銷錯誤操作，是新手工程師必須學會的「救命」技能。

### **5.1 合併衝突的解決**

當Git無法自動將兩個分支的變更合併時，就會發生衝突 23。

* **識別衝突：git status**
  + 當合併操作導致衝突時，git status命令會明確列出所有發生衝突的文件，並提示你這些文件處於「未合併」狀態 23。
* **手動解決衝突：**
  1. **打開衝突文件：** 使用文本編輯器打開衝突文件。你會看到Git插入的特殊衝突標記，例如：  
     <<<<<<< HEAD  
     // 當前分支的修改內容  
     =======  
     // 衝突分支的修改內容  
     >>>>>>> branch\_name  
       
     <<<<<<< HEAD到=======之間是當前分支的內容，=======到>>>>>>> branch\_name之間是衝突分支的內容 25。
  2. **編輯文件：** 根據需求手動修改文件，選擇要保留的內容（可以是當前分支的、衝突分支的，或兩者結合），並**刪除所有衝突標記** 25。
  3. **標記為已解決：git add <file\_name>**
     + 解決完一個文件的衝突後，使用git add <file\_name>命令將修改後的文件添加到暫存區，告訴Git該文件衝突已解決 25。
  4. **完成合併：git commit**
     + 解決所有衝突文件並將它們添加到暫存區後，執行git commit命令完成合併。Git會自動生成一個合併提交訊息，你可以編輯它以說明衝突的解決方式 25。
* **使用合併工具：git mergetool**
  + 對於複雜的衝突，手動解決可能很困難。Git允許你配置外部圖形化合併工具（如Kdiff3、Meld、Beyond Compare）來協助解決衝突 34。
  + **命令：** git mergetool 21。
  + **作用：** 它會為每個衝突文件啟動配置好的合併工具，提供視覺化介面來比較和合併變更 37。

### **5.2 撤銷錯誤操作**

在Git中，撤銷操作有多種方式，但它們的影響範圍和原理各不相同。理解這些差異對於安全地管理程式碼至關重要。

* **撤銷工作區修改：git checkout --, git restore**
  + **git checkout -- <file\_name>：** 將指定文件的工作區修改恢復到最近一次提交的狀態，丟棄所有未暫存的修改 39。
  + **git checkout.：** 撤銷工作區中所有文件的修改 39。
  + **git restore <file\_name> (Git 2.23+ 推薦)：** 作用與git checkout --類似，語義更清晰，專門用於恢復工作區文件 39。
  + **注意：** 這些命令會丟棄工作區中未保存的修改，且無法恢復，使用時務必謹慎 39。
* **撤銷暫存區修改：git reset HEAD, git restore --staged**
  + **git reset HEAD <file\_name>：** 將指定文件從暫存區中移除，但保留工作區的修改。文件狀態從「已暫存」變回「已修改」 39。
  + **git reset HEAD. 或 git reset：** 將所有已暫存的文件從暫存區移除，保留工作區修改 39。
  + **git restore --staged <file\_name> (Git 2.23+ 推薦)：** 作用與git reset HEAD類似，專門用於將文件從暫存區移回工作區 41。
* **撤銷提交：git revert 與 git reset**
  + 這兩者是撤銷已提交變更的常用命令，但原理和適用場景不同。
  + **git revert <commit\_id>：**
    - **原理：** 創建一個**新的提交**，這個新提交的內容會撤銷指定提交所引入的所有變更 43。它不會改變過去的提交歷史，而是新增一條「撤銷」記錄 43。
    - **適用場景：** 常用於撤銷已經推送到遠端倉庫的公開提交 44。由於它不修改歷史，對團隊協作影響最小，是更安全的撤銷方式。
  + **git reset <commit\_id>：**
    - **原理：** 用於回退版本，直接移動HEAD指針和當前分支的指針到指定的提交 44。它會「遺棄」或「刪除」指定提交之後的歷史記錄 44。
    - **參數選項：**
      * --soft：只移動HEAD指針和分支指針，保留工作區和暫存區的內容不變 40。
      * --mixed (預設)：移動HEAD指針和分支指針，並重置暫存區，但保留工作區的內容不變 40。
      * --hard：移動HEAD指針和分支指針，同時重置暫存區和工作區的內容，丟棄所有變更 40。這是最危險的選項，會清除所有未提交的修改，且無法恢復 40。
    - **適用場景：** 常用於撤銷本地尚未推送到遠端倉庫的提交 44。如果已推送到遠端，使用  
      --hard後再git push --force會覆蓋遠端歷史，這對團隊協作是極其危險的操作，應極力避免，除非你完全清楚其後果並已與團隊溝通。
  + **git commit --amend：**
    - **作用：** 用於修改最近一次的提交。如果你剛剛提交的訊息有誤，或想在最近一次提交中添加漏掉的文件，可以使用此命令 18。它會將新的修改和原提交合併成一個新的提交，替換掉舊的提交。
    - **命令：** git commit --amend -m "新的提交訊息" 18。
    - **注意：** 只能修改最近一次的提交，且如果該提交已經推送到遠端，不建議使用，因為這會改變歷史。

## **VI. 良好Git習慣與學習資源**

### **6.1 培養良好Git習慣**

作為新手工程師，培養良好的Git使用習慣，對於個人成長和團隊協作都至關重要 18。

* **頻繁使用git status：** 始終在執行git add或git commit前檢查工作區和暫存區的狀態，避免誤操作 18。
* **提交粒度適中：** 一次提交只包含一個相關的改動（例如一個功能或一個bug修復），避免將多個不相關的變更混在一起。這有助於程式碼審查和未來回溯歷史 17。
* **撰寫清晰有意義的提交訊息：** 提交訊息應簡潔扼要地描述本次提交的內容和目的，這極大地幫助團隊成員理解你的工作，並方便日後查詢 15。
* **提交前進行測試：** 在git commit之前，確保你的程式碼已經過完整的測試，避免提交有問題的程式碼 18。
* **及時推送到遠端：** 在本地倉庫提交到主分支後，盡快推送到遠端倉庫，保持本地與遠端同步，減少合併衝突的機會 18。
* **git pull前先git status：** 在拉取遠端程式碼之前，先查看本地倉庫狀態，確保沒有未提交的變更導致衝突 18。
* **慎用危險命令：** 對於git reset --hard、git rebase等會修改歷史的命令，務必謹慎使用，尤其是在已推送到遠端倉庫的提交上，因為這可能導致團隊成員的程式碼混亂和丟失 18。

### **6.2 推薦學習資源**

Git的學習是一個循序漸進的過程，除了實踐，善用優質的學習資源能事半功倍。

* **官方文檔與書籍：**
  + **Git官方網站：** 提供最權威和詳盡的文檔和指南 47。
  + **《Pro Git》：** 這是一本被廣泛推薦的Git聖經，提供系統的理論知識和深入的內部機制講解，有中文版 10。
* **線上互動式教學平台：**
  + **Learn Git Branching (learngitbranching.js.org)：** 這個網站提供了互動式的學習環境，以類似闖關遊戲的方式，透過命令行視窗和圖形化呈現，幫助你學習Git的基本操作和概念，尤其適合理解分支 48。
  + **Codecademy、W3Schools、菜鳥教程：** 提供Git的基本概念和常用命令的線上教程 47。
* **中文學習資源：**
  + **六角學院 Git & GitHub 教學手冊：** 提供Git與GitHub的教學，包含PR流程等 33。
  + **CSDN、博客園等技術部落格：** 許多資深開發者會分享Git實戰經驗和問題解決方案 47。
  + **CodeLove.tw：** 提供Github Desktop新手入門教學，從專案初始化到連線GitHub帳號，適合初學者 50。
* **推薦書籍（針對新手）：**
  + **《いちばんやさしいGit&GitHubの教本》：** 針對初學者，用簡單的解釋和圖片講解Git和GitHub的基礎，有助於系統性地輸入基本用法和術語 51。
  + **《はじめてでもできる GitとGitHubの教科書》：** 同樣面向初學者，圖文並茂，涵蓋Git基本概念、命令和GitHub專案管理 51。

## **VII. 結論**

掌握Git對於新手工程師而言，不僅是技術能力的提升，更是融入現代軟體開發團隊的必備條件。透過本報告的引導，新手工程師應能理解Git的核心概念，包括其分散式特性、快照式資料處理方式以及工作區、暫存區、儲存庫的三大區域劃分。這些基礎知識是理解Git行為模式的關鍵。

在實際操作層面，從初始化本地儲存庫（git init）開始，到文件修改、暫存（git add）、提交（git commit）的循環，以及透過git status隨時掌握專案狀態，都是日常開發中不可或缺的技能。特別是git status，它就像新手工程師的指南針，能有效降低操作失誤的風險。

分支管理是Git的強大之處，它實現了並行開發和靈活實驗，是團隊協作的基石。學會建立、切換和合併分支，並理解Git Flow和GitHub Flow等分支策略，將使新手工程師能更有效地參與團隊專案。與遠端儲存庫的互動（git clone, git fetch, git pull, git push）則是實現團隊成員間程式碼同步與協作的途徑。

最後，面對程式碼衝突和操作失誤，掌握手動解決衝突的步驟以及選擇合適的撤銷命令（git revert與git reset的區別），是確保程式碼庫健康和維護專案歷史完整性的重要能力。

總而言之，Git的學習是一個實踐的過程。建議新手工程師多加練習，並培養良好的Git使用習慣，如頻繁檢查狀態、撰寫清晰的提交訊息、適度提交粒度等。善用官方文檔和互動式學習平台，將有助於加速學習曲線，最終成為一名能夠熟練運用Git、高效協作的資深工程師。

#### 引用的著作

1. Git核心概念总结| JavaGuide, 檢索日期：7月 2, 2025， <https://javaguide.cn/tools/git/git-intro.html>
2. Git版本控制系统：简介、演变与优缺点原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/cooldream2009/article/details/135122153>
3. Git 的基礎知識 - Nulab, 檢索日期：7月 2, 2025， <https://nulab.com/zh-tw/learn/software-development/git-tutorial/git-basics/>
4. Git简介与工作原理：了解Git的基本概念、版本控制系统和分布式版本 ..., 檢索日期：7月 2, 2025， <https://cloud.tencent.com/developer/article/2405805>
5. Git 的正确使用姿势与最佳实践：团队协作和版本控制的最佳实践｜ 青训营, 檢索日期：7月 2, 2025， <https://juejin.cn/post/7272417917185933324>
6. Git 工作区、暂存区、版本库之间的关系 - Hsu Yeung, 檢索日期：7月 2, 2025， <https://www.hsuyeung.com/article/the-relationship-between-git-workspaces-stage-and-repository>
7. 工作區、暫存區與儲存庫 - 為你自己學Git, 檢索日期：7月 2, 2025， <https://gitbook.tw/chapters/using-git/working-staging-and-repository>
8. Git 对象 - Git, 檢索日期：7月 2, 2025， <https://git-scm.com/book/zh/v2/Git-%E5%86%85%E9%83%A8%E5%8E%9F%E7%90%86-Git-%E5%AF%B9%E8%B1%A1>
9. Git的基本概念/常用命令及实例 - Gitee 帮助中心, 檢索日期：7月 2, 2025， <https://help.gitee.com/enterprise/code-manage/Git%20%E7%9F%A5%E8%AF%86%E5%A4%A7%E5%85%A8/Git%E7%9A%84%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5%E5%92%8C%E5%B8%B8%E7%94%A8%E5%91%BD%E4%BB%A4%E5%8F%8A%E5%AE%9E%E4%BE%8B>
10. Pro Git 中文版（第二版）, 檢索日期：7月 2, 2025， <https://www.progit.cn/>
11. git init详细使用教程原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/weixin_39604653/article/details/141994540>
12. \*\*Git 基礎操作\*\* - HackMD, 檢索日期：7月 2, 2025， <https://hackmd.io/@z20MZE1jSIilXQUk6z8SQQ/ryM2H54F9>
13. git-status – Git コマンドリファレンス（日本語版） - tracpath.com, 檢索日期：7月 2, 2025， <https://tracpath.com/docs/git-status/>
14. git status について - Laravel学習帳, 檢索日期：7月 2, 2025， <https://laraweb.net/environment/4686/>
15. Git Commit Message的最佳实践原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/qq_42445025/article/details/119347006>
16. 【译】编写GIT 提交信息的最佳实践 - 稀土掘金, 檢索日期：7月 2, 2025， <https://juejin.cn/post/6921909971315589133>
17. コミット粒度とコミット名で、レビュー時間は1/3になる - Zenn, 檢索日期：7月 2, 2025， <https://zenn.dev/tkydev/articles/4286cb9691dd6b>
18. 使用git的一些原则、好习惯和常用命令原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/u013553529/article/details/88361442>
19. git团队协作分支• Worktile社区, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/265175.html>
20. 团队合作利器：Git分支详解- 心脏鼓点 - 博客园, 檢索日期：7月 2, 2025， <https://www.cnblogs.com/TD1900/p/13920490.html>
21. Git 命令- 分支与合并, 檢索日期：7月 2, 2025， <https://git-scm.com/book/zh/v2/%E9%99%84%E5%BD%95-C:-Git-%E5%91%BD%E4%BB%A4-%E5%88%86%E6%94%AF%E4%B8%8E%E5%90%88%E5%B9%B6>
22. Git 的分支指令 - Nulab, 檢索日期：7月 2, 2025， <https://nulab.com/zh-tw/learn/software-development/git-tutorial/git-commands-settings/git-branch-commands/>
23. 解決Git 合併衝突- Azure Repos | Microsoft Learn, 檢索日期：7月 2, 2025， <https://learn.microsoft.com/zh-tw/azure/devops/repos/git/merging?view=azure-devops>
24. 團隊協作Git Flow. 今天來講一個對於開發也非常重要的一環 - Engine Bai, 檢索日期：7月 2, 2025， <https://enginebai.medium.com/git-flow-60b9466e9942>
25. 再也不怕Git冲突啦（保姆级教学超详细图文版） 原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/weixin_49708955/article/details/130570666>
26. GitHub Flow 策略的优缺点- AWS 规范性指导, 檢索日期：7月 2, 2025， <https://docs.aws.amazon.com/zh_cn/prescriptive-guidance/latest/choosing-git-branch-approach/advantages-and-disadvantages-of-the-git-hub-flow-strategy.html>
27. GitFlow工作流简化版流程图模板 - ProcessOn, 檢索日期：7月 2, 2025， <https://www.processon.com/view/6122fe4a63768958e5317159>
28. 小規模開発チームのブランチ戦略。GitHub Flowの導入 - Insight Edge Tech Blog, 檢索日期：7月 2, 2025， <https://techblog.insightedge.jp/entry/branch-strategy-github-flow>
29. GitHub Flow (Japanese translation), 檢索日期：7月 2, 2025， <https://gist.github.com/Gab-km/3705015>
30. 第25 天：使用GitHub 遠端儲存庫- 觀念篇, 檢索日期：7月 2, 2025， <https://github.com/doggy8088/Learn-Git-in-30-days/blob/master/zh-tw/25.md>
31. 如何使用Git 进行多人协作开发（全流程图解） 原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/whc18858/article/details/133209975>
32. Remote 遠端協同工作- remote, fetch, pull, push | Johnny Wang Blog, 檢索日期：7月 2, 2025， <https://johnnywang1994.github.io/book/articles/git/remote.html>
33. GitHub 發PR (pull request) 流程- Git 遠端多人協作| W3HexSchool - 六角學院, 檢索日期：7月 2, 2025， <https://w3c.hexschool.com/git/cc7d70b7>
34. 解决git冲突的最佳命令 - Worktile, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/254793.html>
35. git如何解决冲突最安全 - Worktile, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/220601.html>
36. 在GitHub 上解决合并冲突, 檢索日期：7月 2, 2025， <https://docs.github.com/zh/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-on-github>
37. git-mergetool - 运行合并冲突解决工具以解决合并冲突, 檢索日期：7月 2, 2025， <https://git-scm.com/docs/git-mergetool/zh_HANS-CN>
38. 名称 - Git - git-merge Documentation, 檢索日期：7月 2, 2025， <https://git-scm.com/docs/git-merge/zh_HANS-CN>
39. git 如何取消工作区的修改 - Worktile, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/220422.html>
40. git 撤销修改和版本回退- 一个人的孤独自白 - 博客园, 檢索日期：7月 2, 2025， <https://www.cnblogs.com/-mrl/p/15476877.html>
41. git如何去掉暂存区修改 - Worktile, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/221541.html>
42. git如何撤销暂存区修改 - Worktile, 檢索日期：7月 2, 2025， <https://worktile.com/kb/ask/221313.html>
43. 『现学现忘』Git后悔药— 32、revert撤销（一） - 繁华似锦Fighting - 博客园, 檢索日期：7月 2, 2025， <https://www.cnblogs.com/liuyuelinfighting/p/16788088.html>
44. 面试官：说说你对git reset 和git revert 的理解？区别？ - Vue3, 檢索日期：7月 2, 2025， <https://vue3js.cn/interview/git/git%20reset_%20git%20revert.html>
45. 面试官：说说你对git reset 和git revert 的理解？区别？ #230 - GitHub, 檢索日期：7月 2, 2025， <https://github.com/febobo/web-interview/issues/230>
46. [レポート] 『きれいなcommit, pull requestを知りたい/作りたい方のためのgit勉強会』に参加してきました | DevelopersIO, 檢索日期：7月 2, 2025， <https://dev.classmethod.jp/articles/git-workshop-20180327/>
47. git学习资源原创 - CSDN博客, 檢索日期：7月 2, 2025， <https://blog.csdn.net/weixin_55887103/article/details/148877907>
48. #65 Git 練習網站介紹— Learn Git Branching | by Jason Hung | 彼得潘的Swift iOS / Flutter App 開發教室| Medium, 檢索日期：7月 2, 2025， <https://medium.com/%E5%BD%BC%E5%BE%97%E6%BD%98%E7%9A%84-swift-ios-app-%E9%96%8B%E7%99%BC%E6%95%99%E5%AE%A4/65-git-%E7%B7%B4%E7%BF%92%E7%B6%B2%E7%AB%99%E4%BB%8B%E7%B4%B9-learn-git-branching-c46c4b66cc2b>
49. 最佳互動式瀏覽器教學: r/git - Reddit, 檢索日期：7月 2, 2025， <https://www.reddit.com/r/git/comments/14g00x1/best_interactive_browser_based_tutorial/?tl=zh-hant>
50. Github Desktop 新手入門教學—— 用漂亮的GUI 介面來學習Git 基本觀念 - CodeLove 論壇, 檢索日期：7月 2, 2025， <https://codelove.tw/@howtomakeaturn/course/yawb3R>
51. 【入門から応用】Git/Githubの学習におすすめの本/書籍5選｜webdrawer - note, 檢索日期：7月 2, 2025， <https://note.com/webdrawer/n/nfe6fdc41e56a>