Git Commands E-Book

By Filipe V. Moreno

Principais comandos:

1. `git init`:
- Funcionalidade: Inicializa um novo repositório Git no diretório atual, criando um novo repositório vazio.

2. `git clone <URL_do_repositório>`:
- Funcionalidade: Clona um repositório Git existente para o diretório atual, criando uma cópia local do repositório remoto.

3. `git add <arquivo(s)>`:
- Funcionalidade: Adiciona arquivos ou mudanças específicas para serem preparados para o próximo commit.

4. `git commit -m "Mensagem de commit"`:
- Funcionalidade: Registra as mudanças preparadas (adicionadas com `git add`) em um novo commit com uma mensagem descritiva.

5. `git status`:
- Funcionalidade: Mostra o status atual das mudanças no repositório, indicando quais arquivos estão modificados, adicionados ou removidos.

6. `git log`:
- Funcionalidade: Exibe um histórico de commits do repositório, mostrando informações como o autor, a data e a mensagem de cada commit.

7. `git branch`:
- Funcionalidade: Lista todas as branches (ramificações) no repositório e indica qual branch está atualmente em uso.

8. `git checkout <nome_da_branch>`:
- Funcionalidade: Muda para outra branch, permitindo que você trabalhe em diferentes ramificações do seu projeto.

9. `git pull`:
- Funcionalidade: Atualiza o repositório local com as mudanças do repositório remoto, trazendo as atualizações da branch atual.

10. `git push`:
- Funcionalidade: Envia as mudanças locais para o repositório remoto, atualizando a branch correspondente no servidor.

11. `git merge <nome_da_branch>`:
- Funcionalidade: Combina as mudanças de outra branch na branch atual, criando um novo commit de mesclagem.

12. `git remote -v`:
- Funcionalidade: Mostra os repositórios remotos vinculados ao repositório local e suas URLs.

13. `git reset <arquivo>`:
- Funcionalidade: Remove um arquivo da área de preparação (staging) após ter sido adicionado com `git add`, desfazendo a mudança preparada.

14. `git stash`:
- Funcionalidade: Guarda temporariamente as mudanças locais em um stash, permitindo que você trabalhe em outra tarefa antes de aplicar as mudanças de volta.

15. `git remote add <nome_do_repositório> <URL_do_repositório>`:
- Funcionalidade: Adiciona um novo repositório remoto com um nome específico para facilitar futuras operações de push e pull.

16. `git fetch`:
- Funcionalidade: Recupera as atualizações do repositório remoto, mas não aplica automaticamente essas atualizações à sua branch local. É útil para verificar se há mudanças sem mesclá-las imediatamente.

17. `git branch -d <nome_da_branch>`:
- Funcionalidade: Exclui uma branch local após ter sido mesclada com sucesso em outra branch. Use `-D` em vez de `-d` para forçar a exclusão, mesmo se a branch não tiver sido mesclada.

18. `git diff`:
- Funcionalidade: Mostra as diferenças entre os arquivos no diretório de trabalho e o último commit, permitindo que você veja as mudanças antes de prepará-las com `git add`.

19. `git tag <nome_da_tag>`:
- Funcionalidade: Cria uma tag para marcar um ponto específico na história do repositório, geralmente usado para marcar versões estáveis do software.

20. `git rebase <branch_alvo>`:

- Funcionalidade: Move ou "rebaseia" os commits da branch atual sobre a branch especificada, resultando em uma linha do tempo mais linear. Útil para manter um histórico de commits limpo.

Comandos intermediários:

21. `git remote remove <nome_do_repositório>`:

- Funcionalidade: Remove um repositório remoto previamente vinculado ao repositório local.

22. `git clean -n` e `git clean -f`:

- Funcionalidade: `-n` mostra quais arquivos não rastreados seriam removidos com o comando `-f` (força) realmente os remove. Cuidado ao usar `-f`, pois ele apaga permanentemente os arquivos não rastreados.

23. `git blame <arquivo>`:

- Funcionalidade: Mostra quem fez cada alteração em cada linha de um arquivo, útil para rastrear a origem de alterações específicas.

24. `git cherry-pick <commit>`:

- Funcionalidade: Aplica um commit específico de outra branch na branch atual, permitindo que você escolha commits individuais para mesclar.

25. `git bisect`:

- Funcionalidade: Ajuda a encontrar o commit que introduziu um bug, automatizando um processo de pesquisa binária para identificar o commit problemático.

26. `git cherry-pick -x <commit>`:

- Funcionalidade: Similar ao `git cherry-pick`, mas com a opção `-x`, ele inclui automaticamente uma linha na mensagem do commit indicando de onde a mesclagem foi realizada.

27. `git reflog`:

- Funcionalidade: Exibe um registro de todas as ações que afetaram a posição da ramificação HEAD, útil para recuperar commits perdidos ou desfazer operações acidentais.

28. `git submodule`:

- Funcionalidade: Gerencia submódulos em um repositório Git. Submódulos permitem incluir repositórios Git dentro de outro repositório, útil para gerenciar dependências.

29. `git filter-branch`:

- Funcionalidade: Permite reescrever o histórico de um repositório Git, útil para realizar tarefas de limpeza e reestruturação em um histórico de commits.

30. `git rebase -i <branch>`:

- Funcionalidade: Inicia uma rebase interativa, que permite reorganizar, editar ou mesclar commits enquanto estão sendo rebased.

Comandos avançados:

31. `git blame -L <linha-inicial>,<linha-final> <arquivo>`:

- Funcionalidade: Mostra quem fez as alterações em um intervalo específico de linhas de um arquivo, útil para rastrear mudanças específicas.

32. `git remote prune origin`:

- Funcionalidade: Remove referências locais para branches remotas que foram excluídas no repositório remoto, mantendo seu repositório local limpo.

33. `git revert <commit>`:

- Funcionalidade: Cria um novo commit que desfaz as mudanças introduzidas por um commit específico, permitindo a reversão de alterações indesejadas.

34. `git bisect` (comando completo):

- Funcionalidade: Um processo de busca binária para encontrar o commit que introduziu um bug. Envolve os comandos `git bisect start`, `git bisect bad`, `git bisect good`, e assim por diante.

35. `git archive`:

- Funcionalidade: Cria um arquivo compactado contendo o conteúdo de uma branch ou commit específico, geralmente usado para criar distribuições de código.

36. `git clean -i`:

- Funcionalidade: Inicia uma limpeza interativa que permite revisar e selecionar os arquivos não rastreados que você deseja remover.

Considerações finais:

Esta pequena coletânea de comandos, não apresenta toda a totalidade do git, apenas demonstra de maneira rápida e transparente os principais comandos utilizados no dia a dia. Leve em conta que estes comandos foram retirados da documentação, e da minha utilização pessoal, assim, sua classificação sendo variável e altamente enviesada ao meu uso, todavia serve como um bom dicionário para rápido entendimento e um bom agrupamento de informações para pessoas que queiram aprender de novos comandos de maneira direta.